TRAVERSIERUNGSVARIANTEN

Verwandte Besuchen: Wie, wann und wo ein gutes Kind zu den geliebten Eltern rennt.

Florian Sihler

13. Juli 2022
SP, Universität Ulm

TRAVERSIERUNGSVARIANTEN

Verwandte Besuchen: Wie, wann und wo ein gutes Kind zu den geliebten Eltern rennt.

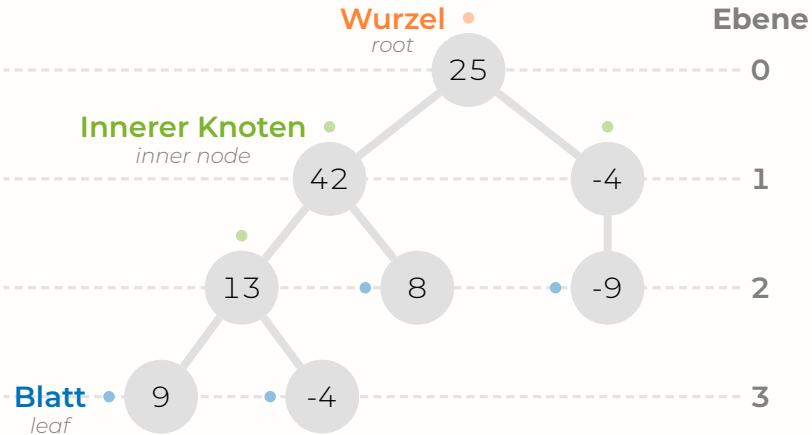
Florian Sihler

Officially supported by the Ping-to-the-u-Foundation for Emotional Support. Look out for others so they may waddle with you!

13. Juli 2022
SP, Universität Ulm

Der binäre Baum

Zur Wiederholung:
Blätter haben keine Kindknoten.

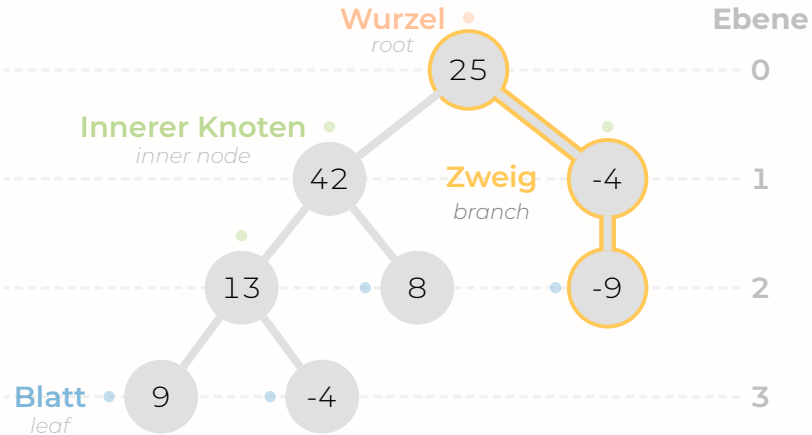


[4]: *LaTeX-Package, tikzpingus* Sihler, 2021

[5]: *Episode-Heaps* Sihler, 2021

Der binäre Baum

Die gilt auch für Bäume
mit mehr als zwei Kindern.

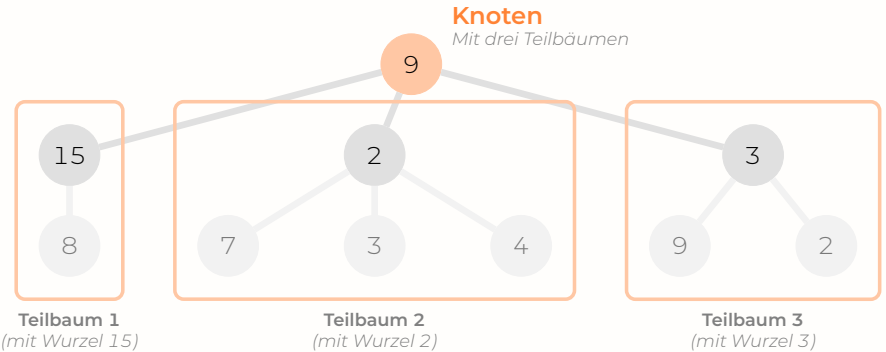


[4]: *LaTeX-Package, tikzpingus* Sihler, 2021

[5]: *Episode-Heaps* Sihler, 2021

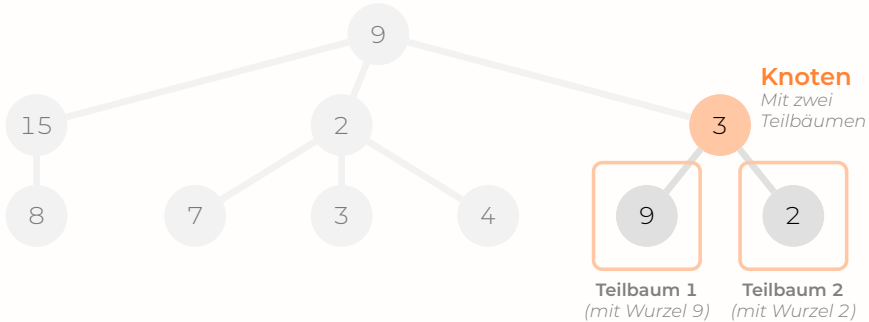
Der rekursive Baum

Jeder Knoten ist Wurzel eines Teilbaums



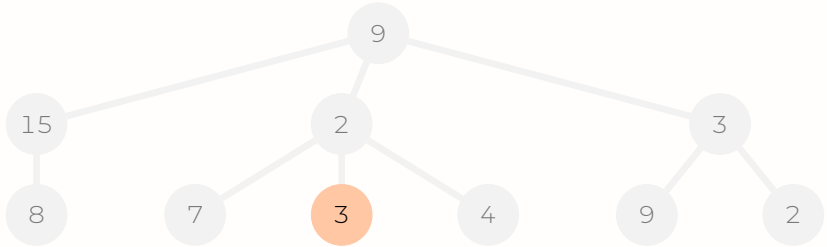
Der rekursive Baum

So lassen sich rekursive Regeln formulieren!



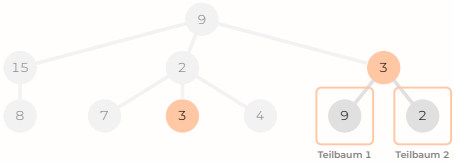
Der rekursive Baum

Beispielsweise:
Left < Root ≤ Right
Root = Left + Right



Knoten
Ohne Teilbäume

Der rekursive Baum

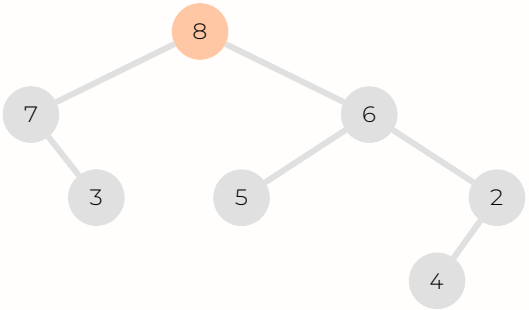


- Binärbäume: *linker* und *rechter* Teilbaum
- *Fehlende* Teilbäume heißen auch *leer*
- Die Betrachtung wird in Java benutzt:

```
class Node {
    int value;
    Node leftSubtree, rightSubtree;
}
```

4 Level-Order (Breadth-First)

Ebene für Ebene, von links nach rechts.



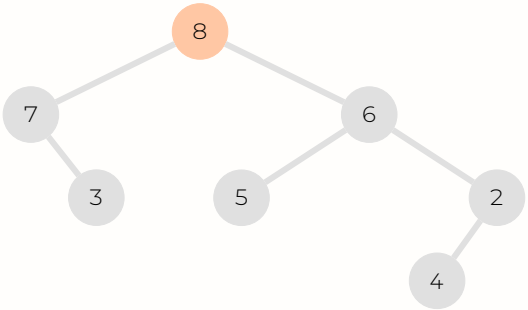
8, 7, 6, 3, 5, 2, 4

[2]: *Depth-first and breadth-first search* Kozen, 1992

Pre-Order (Depth-First)

Allgemeiner: Erst
Wurzel, dann Kinder

Erst *Wurzel*, dann Links, dann Rechts

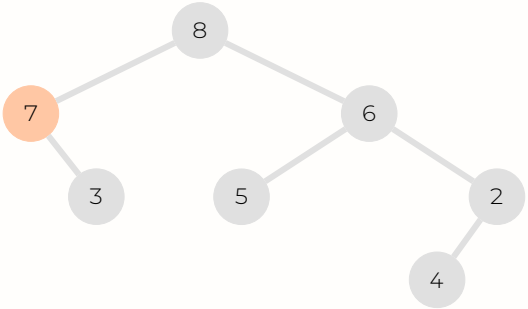


8, 7, 3, 6, 5, 2, 4

[2]: *Depth-first and breadth-first search* Kozen, 1992

6 In-Order (Depth-First)

Erst Links, dann *Wurzel*, dann Rechts



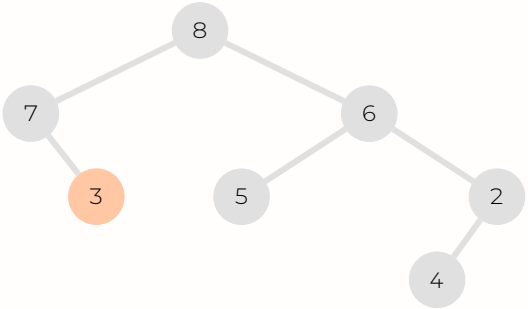
7, 3, 8, 5, 6, 4, 2

[2]: *Depth-first and breadth-first search* Kozen, 1992

Post-Order (Depth-First)

Allgemeiner: Erst Kinder, dann Wurzel

Erst Links, dann Rechts, dann *Wurzel*



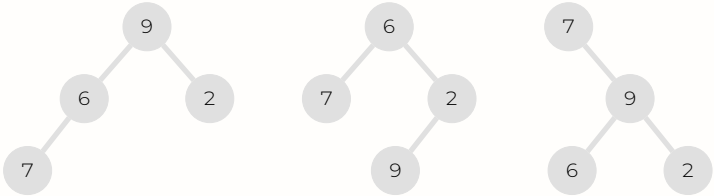
3, 7, 5, 4, 2, 6, 8

[2]: *Depth-first and breadth-first search* Kozen, 1992

Kommentare

- Die Reihenfolge ist *keine* eindeutige Beschreibung!

In-Order: 7, 6, 9, 2

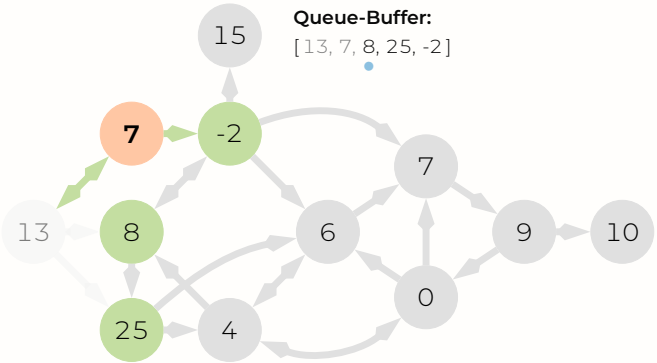


Kommentare

- Es gibt auch „reverse“-Varianten.
  Diese bearbeiten den rechten *vor* dem linken Teilbaum
- Es werden Breiten- und Tiefendurchläufe unterschieden:
  - *depth-first*:
    Bearbeitet Pfad komplett und dann Abzweigungen
  - *breadth-first*:
    Breitet sich über alle Abzweigungen gleichmäßig aus

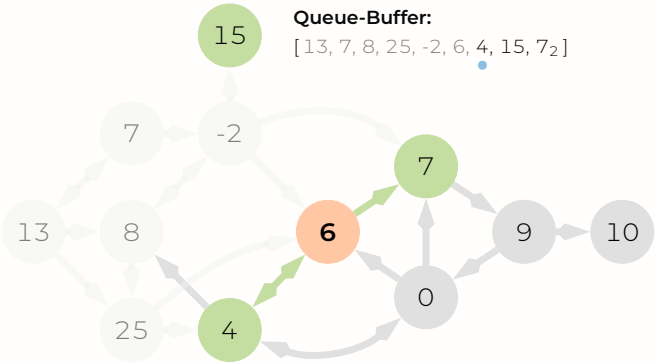
10.1 Graphen in der Breite

- **Breiten-** und Tiefendurchläufe gehen auch auf Graphen!



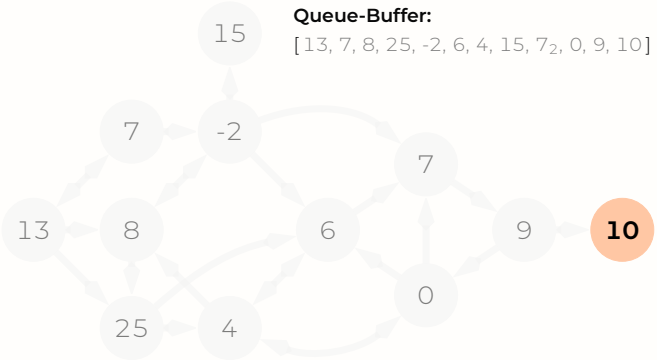
10.2 Graphen in der Breite

- **Breiten-** und Tiefendurchläufe gehen auch auf Graphen!



10.3 Graphen in der Breite

- **Breiten-** und Tiefendurchläufe gehen auch auf Graphen!

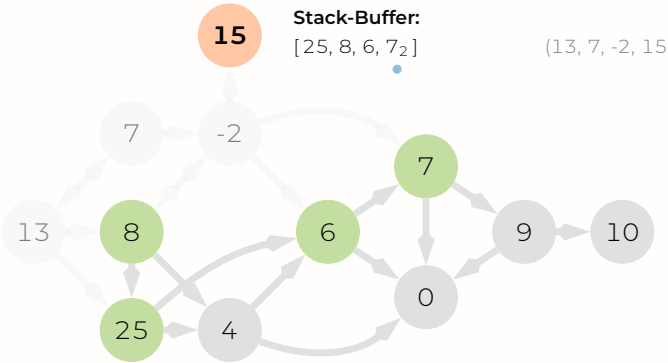


Queue-Buffer:
[13, 7, 8, 25, -2, 6, 4, 15, 7_2, 0, 9, 10]

Graphen in der Tiefe

Achtung: Zur Anschaulichkeit habe ich ein paar Kanten abgeändert!

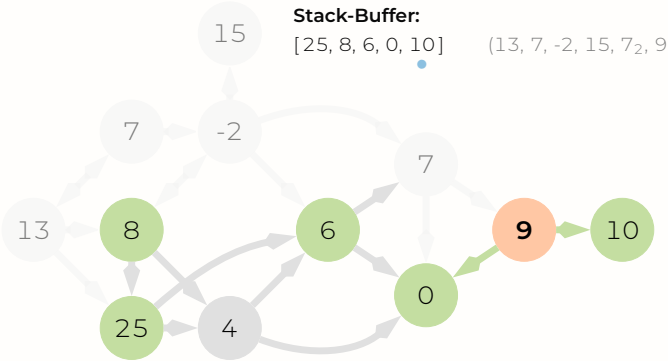
- Breiten- und **Tiefendurchläufe** gehen auch auf Graphen!



Graphen in der Tiefe

Achtung: Zur Anschaulichkeit habe ich ein paar Kanten abgeändert!

- Breiten- und **Tiefendurchläufe** gehen auch auf Graphen!



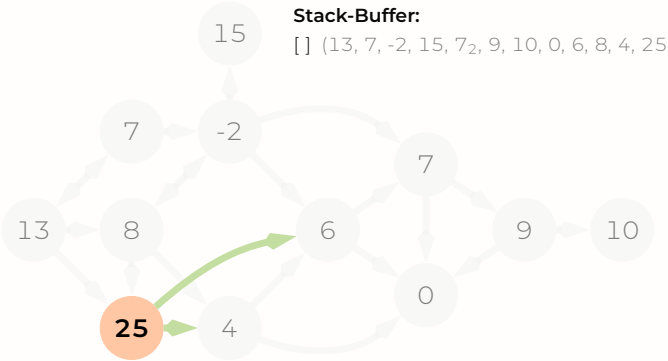
Stack-Buffer:
[25, 8, 6, 0, 10] (13, 7, -2, 15, 7_2, 9)

Die Reihenfolge, mit der die Kinder auf den Stack kommen, hängt natürlich von der Implementation ab!

Graphen in der Tiefe

Achtung: Zur Anschaulichkeit habe ich ein paar Kanten abgeändert!

- Breiten- und **Tiefendurchläufe** gehen auch auf Graphen!



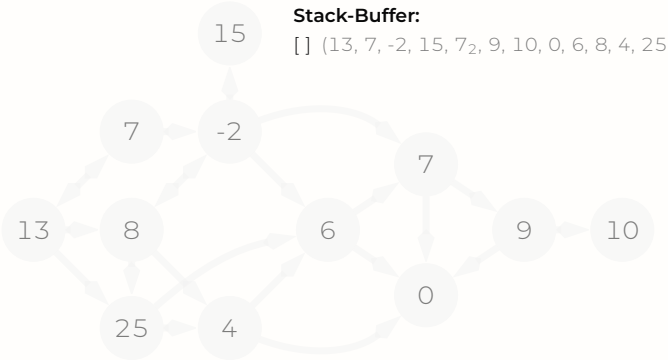
Stack-Buffer:
[ ] (13, 7, -2, 15, 7_2, 9, 10, 0, 6, 8, 4, 25)

Die Reihenfolge, mit der die Kinder auf den Stack kommen, hängt natürlich von der Implementation ab!

Graphen in der Tiefe

Achtung: Zur Anschaulichkeit habe ich ein paar Kanten abgeändert!

- Breiten- und **Tiefendurchläufe** gehen auch auf Graphen!



Stack-Buffer:
[ ] (13, 7, -2, 15, 7_2, 9, 10, 0, 6, 8, 4, 25)

Die Reihenfolge, mit der die Kinder auf den Stack kommen, hängt natürlich von der Implementation ab!

12.1 Irrgärten

- Ein Labyrinth ist auch nur ein cooler Graph

		1	3	5	8		17	19	
	2		6	9			15		
	4			11	12	13	14		
	7	10					16		
				23	22	21	18	20	

Bei dieser beispielhaften Breitensuche darf der Läufer nicht diagonal gehen!

12.2 Irrgärten

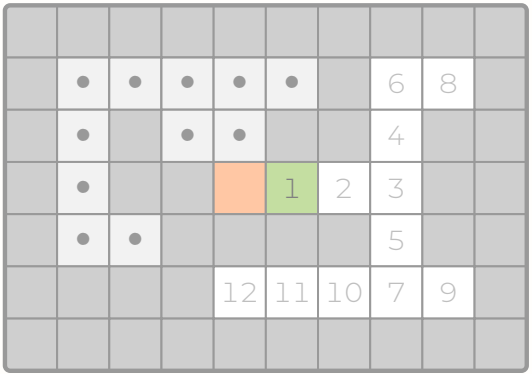
- Ein Labyrinth ist auch nur ein cooler Graph

	•	•	•	•	1		10	12	
	•		•	2			8		
	•			4	5	6	7		
		3					9		
				16	15	14	11	13	

Bei dieser beispielhaften Breitensuche darf der Läufer nicht diagonal gehen!

12.3 Irrgärten

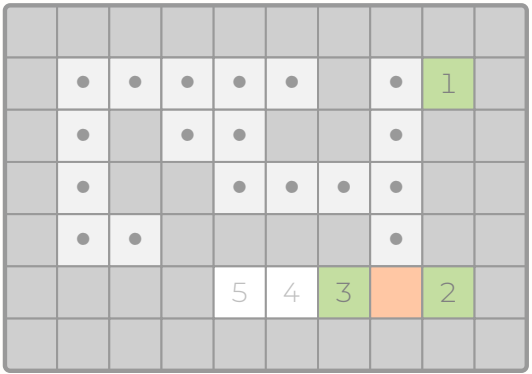
- Ein Labyrinth ist auch nur ein cooler Graph



Bei dieser beispielhaften Breitensuche darf der Läufer nicht diagonal gehen!

12.4 Irrgärten

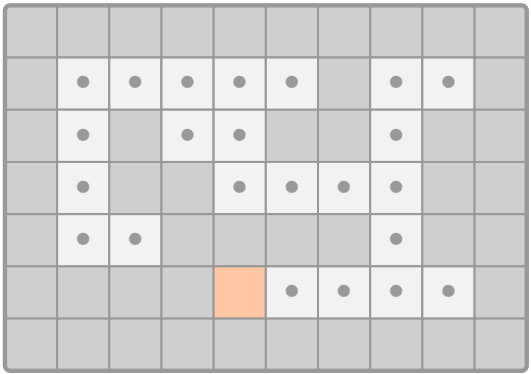
- Ein Labyrinth ist auch nur ein cooler Graph



Bei dieser beispielhaften Breitensuche darf der Läufer nicht diagonal gehen!

12.5 Irrgärten

- Ein Labyrinth ist auch nur ein cooler Graph



Bei dieser beispielhaften Breitensuche darf der Läufer nicht diagonal gehen!

Ein Breitendurchlauf

- Ein einfaches „Labyrinth“ und viel AHA:

		1	4	9	16	20	22	24	
	3	2	5	10					
	8	7	6	11					
	15	14	13	12	17	21	23	25	
	19			18					

Hier darf der Läufer diagonal gehen!

13.2 Ein Breitendurchlauf

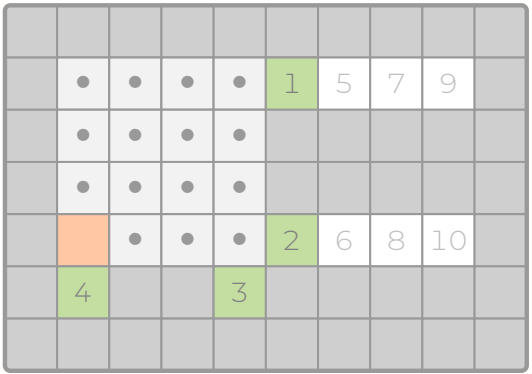
- Ein einfaches „Labyrinth“ und viel AHA:

	•	•	•		7	11	13	15	
	•	•	•	1					
	•	•	•	2					
	6	5	4	3	8	12	14	16	
	10			9					

Hier darf der Läufer diagonal gehen!

13.3 Ein Breitendurchlauf

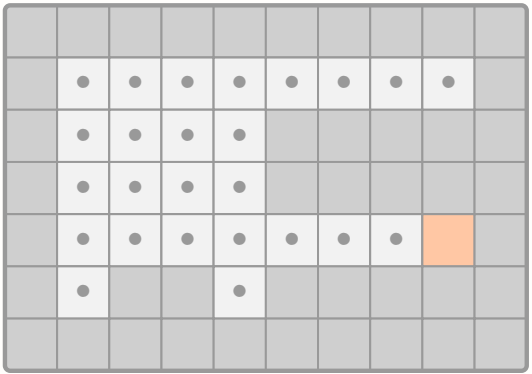
- Ein einfaches „Labyrinth“ und viel AHA:



Hier darf der Läufer diagonal gehen!

Ein Breitendurchlauf

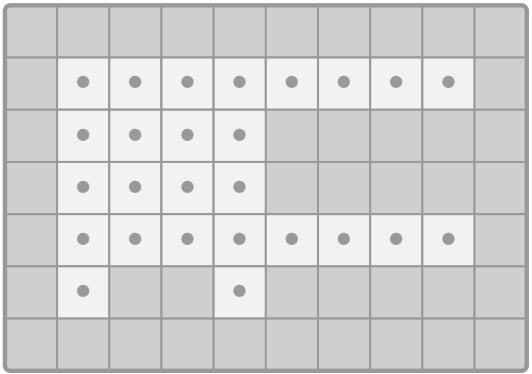
- Ein einfaches „Labyrinth“ und viel AHA:



Hier darf der Läufer diagonal gehen!

Ein Breitendurchlauf

- Ein einfaches „Labyrinth“ und viel AHA:



Hier darf der Läufer diagonal gehen!

Ein Tiefendurchlauf

- Ein einfaches „Labyrinth“ und viel AHA:

		1	2	3	4	5	6	7	
	21	22	23	8					
	20	25	24	9					
	18	17	16	10	12	13	14	15	
	19			11					

Hier darf der Läufer diagonal gehen!

Ein Tiefendurchlauf

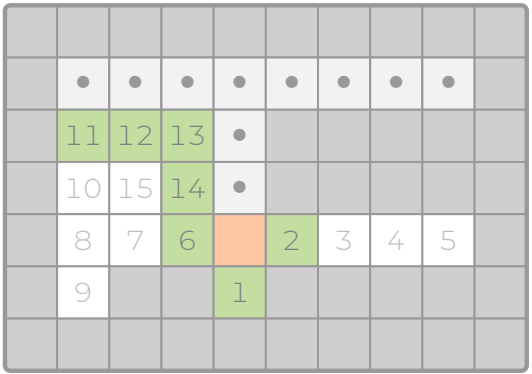
- Ein einfaches „Labyrinth“ und viel AHA:

	•	•	•	•	•	•	•		
	14	15	16	1					
	13	18	17	2					
	11	10	9	3	5	6	7	8	
	12			4					

Hier darf der Läufer diagonal gehen!

Ein Tiefendurchlauf

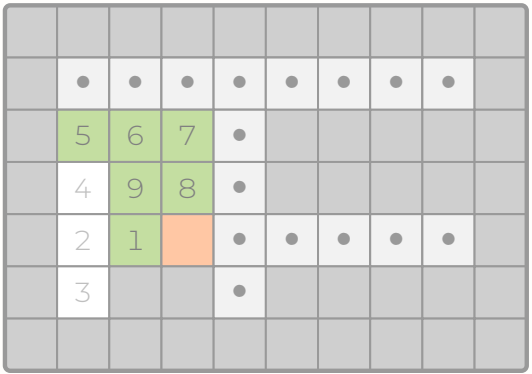
- Ein einfaches „Labyrinth“ und viel AHA:



Hier darf der Läufer diagonal gehen!

Ein Tiefendurchlauf

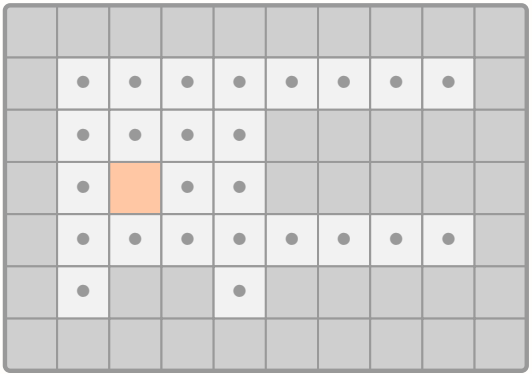
- Ein einfaches „Labyrinth“ und viel AHA:



Hier darf der Läufer diagonal gehen!

Ein Tiefendurchlauf

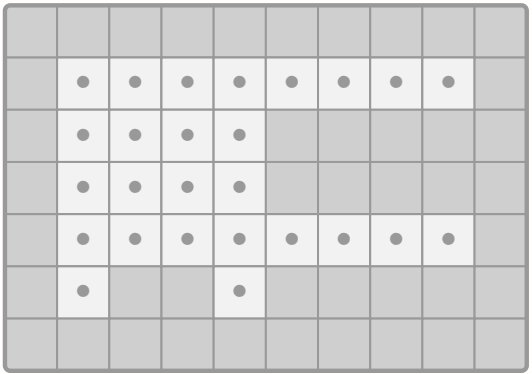
- Ein einfaches „Labyrinth“ und viel AHA:



Hier darf der Läufer diagonal gehen!

Ein Tiefendurchlauf

- Ein einfaches „Labyrinth“ und viel AHA:



Hier darf der Läufer diagonal gehen!

Abschließendes

- Dies war ein primär visueller Blick auf Traversierungen
- Es gibt vieeel mehr (z.B. die „Wirbeltraversierung“)
- Die Kenntnis über die iterative und rekursive Implementation ist nützlich!
- Breiten- & Tiefensuche helfen auch bei der Wegfindung

[3]: *Scanning list structures without stacks or tag bits* Lindstrom, 1973

[6]: *Software solution for optimal planning of sales persons work based on Depth-First Search and Breadth-First Search algorithms* Žunić, 2016

[1]: *Pathfinding of 2D & 3D game real-time strategy with depth direction A* algorithm for multi-layer* Khantanapoka, 2009

[1] Khammapun Khantanapoka und Krisana Chinnasarn. „Pathfinding of 2D & 3D game real-time strategy with depth direction A* algorithm for multi-layer“. 2009

[2] Dexter C Kozen. „Depth-first and breadth-first search“. 1992

[3] Gary Lindstrom. „Scanning list structures without stacks or tag bits“. 1973

[4] Florian Sihler. *LaTeX-Package, tikzpingus*. 2021

[5] Florian Sihler. *Episode-Heaps*. 2021

[6] E. Žunić, A. Djedović und B. Žunić. „Software solution for optimal planning of sales persons work based on Depth-First Search and Breadth-First Search algorithms“. 2016

Florian Sihler
Ulm, 13. Juli 2022

Slides and LaTeX-sources on GitHub!