

Algorithmen und Datenstrukturen

Lecture 1

Prof. Dr. Maximilian Scherer
maximilian.scherer@dhw.de



SoSe 2025

Herzlich Willkommen

- ▶ Modul Fortgeschrittene Programmierung
- ▶ Einheit **Algorithmen und Datenstrukturen**

Vorstellungsrunde

Ihr Name und ein uninteressanter Aspekt zu Ihrer Person

Ziel der Veranstaltung

- ① Befähigung zum algorithmischen Denken
- ② Algorithmen und Datenstrukturen verstehen und implementieren
- ③ Effizienz (Laufzeit und Speicherbedarf) analysieren

Organisatorisches

- ▶ Vorlesung (30 VE)
- ▶ Selbststudium (45 VE)
 - Nachbereitung
 - Vorbereitung: Fragen / Notizen zu Thema
 - Übungsaufgaben (Bonuspunkte)
 - Klausurvorbereitung

Prüfungsleistung

Modul mit 120 Punkten

- ▶ Klausurteil mit 60 Punkten
- ▶ Bonuspunkte für die Klausur (nur Teil Algo & DS)

Bonuspunkte

- ▶ Bis zu 5 Bonuspunkte für die Klausur
- ▶ Klausurpunkte + Bonus maximal 60 Punkte
- ▶ In Veranstaltung 1 bis 5 je eine Übungsaufgabe
- ▶ Moodle Abgabe bis **19 Uhr vor** der nächsten Veranstaltung
- ▶ Wer abgibt, muss vorstellen können (sonst Disqualifikation vom Bonusprogramm)

Bonuspunkte

- ▶ <https://leetcode.com/problemset/all/>
- ▶ Gruppenarbeit möglich (1-3 Studierende)
- ▶ Individuelle Moodle Abgabe
- ▶ Abgabe: Code + Screenshot in zwei Dateien
 - **txt/py** Datei mit Name, Vorname, Matrikel (für alle Gruppenmitglieder) und Quellcode (Programmiersprache egal)
 - **png/jpg** Screenshot mit Submission Result und Codeausschnitt
- ▶ 1 Punkt pro funktionsfähiger Abgabe (Laufzeit egal)

[Explore](#)[Problems](#)[Interview](#)**New**[Contest](#)[Discuss](#)[Store](#)[Description](#)[Solution](#)[Discuss \(999+\)](#)[Submissions](#)

Python3

 Autocomplete**Success** [Details >](#)

Runtime: **168 ms**, faster than **92.34%** of Python3 online submissions for Majority Element.

Memory Usage: **15.6 MB**, less than **32.74%** of Python3 online submissions for Majority Element.

```
1 from collections import Counter
2 class Solution:
3     def majorityElement(self, nums: List[int]) -> int:
4         c = Counter(nums)
5         return c.most_common(1)[0][0]
```

Klausur

- ▶ Spickzettel / Cheat-Sheet Klausur
 - DIN A4 doppelseitig **handgeschrieben**
 - Pflichtinhalt: lesbare Matrikelnummer
- ▶ Klausur:
 - 2 Leetcode Aufgaben (1 bekannt (abgeändert, ähnlich), 1 neu)
 - Lösungsidee als Fließtext
 - Python/Pseudocode mit Laufzeit / Speicherbedarf Analyse

Moodle-Raum

- ▶ Moodle Kursraum:

<https://moodle.dhbw-mannheim.de/course/view.php?id=12991>

- ▶ Schlüssel: 124

Intro to Algo & DS

Literatur / Referenzen

- ▶ The Algorithm Design Manual by Steven S. Skiena

<https://www.algorist.com/>¹

- ▶ Introduction to Algorithms by Thomas H. Cormen²

- ▶ Intro to Algorithms, Erik Demaine MIT

<https://www.youtube.com/playlist?list=PLU14u3cNGP63EdVPNLG3ToM6LaEUuStEY>

¹Steven S. Skiena. **The Algorithm Design Manual**. 2nd. Springer Publishing Company, Incorporated, 2008. ISBN: 1848000693.

²Thomas H. Cormen et al. **Introduction to Algorithms, Third Edition**. 3rd. The MIT Press, 2009. ISBN: 0262033844.

- ▶ Was ist ein Algorithmus?
- ▶ Endliche Sequenz wohldefinierter Instruktionen
- ▶ Beispiele
 - Berechnen / Arithmetik
 - Suchen
 - Sortieren
 - Planen
 - Simulieren
 - Visualisieren und Interagieren

- ▶ Was ist eine Datenstruktur?
- ▶ organization, management, and storage format
- ▶ auf effizienten Zugriff optimiert
- ▶ Beispiel: Integer als Zeichenkette speichern. Effizient?

► Beispiele für Datenstrukturen

- Primitive
- Arrays / Listen
- Queues / Stacks
- Dictionaries
- Bäume
- Graphen

Anforderungen an einen Algorithmus:

- ▶ Endlich
- ▶ Korrekt
- ▶ Effizient (Laufzeit und Speicher)

Spezifikation eines Algorithmus:

- ▶ Sprachlich
- ▶ Diagramme
- ▶ Pseudo Code
- ▶ Programm Code
- ▶ Maschinen Code
- ▶ elektronische Schaltung

Spezifikation eines Algorithmus (was wir behandeln):

- ▶ **Sprachlich**
- ▶ Diagramme
- ▶ **Pseudo Code**
- ▶ **Programm Code**
- ▶ Maschinen Code
- ▶ elektronische Schaltung

Ein erster Beispiel

Maximum Element

Gegeben ein Array X mit N Integer Elementen. Was ist das größte Element?

- ① Nehme an, das erste Element ist das Maximum
- ② Vergleiche das nächste Element des Arrays mit dem bisherigen Maximum.
- ③ Wenn es größer ist, aktualisiere das Maximum auf dieses Element
- ④ Wiederhole 2 und 3 bis wir am Ende des Arrays angelangt sind

Maximum Element

```
1 MaxElement(int[] X):  
2     maxVal = X[0]  
3     foreach nextVal in X:  
4         if nextVal > maxVal:  
5             maxVal = nextVal  
6     return maxVal
```

- ▶ Endlich? Die For-Schleife iteriert jedes Eingabeelement genau einmal und ist damit endlich.
- ▶ Korrekt? Beweis per Induktion / Schleifeninvariante
- ▶ Speicher / Laufzeit? $\mathcal{O}(1)$ Speicher (kein extra Speicher; unabhängig von Eingabe), $\mathcal{O}(n)$ Laufzeit (n Vergleiche und maximal n Zuweisungen)

Fokus dieser Veranstaltung

- ▶ Ideen und Konzepte für Algorithmen / Datenstrukturen
- ▶ Algorithmen / Datenstrukturen als Pseudo Code entwerfen
- ▶ Effizienz analysieren
- ▶ Praktisch implementieren
- ▶ Keine formalen Beweise bzgl. Endlichkeit und Korrektheit

Datenstruktur: Statisches Array

- ▶ Was ist ein Array?
- ▶ statischer, zusammenhängender Speicherbereich fester Blockgröße
- ▶ `myArray = int[5] {1,2,3,9,-5}`
- ▶ Speicherbedarf für 5 Elemente?
- ▶ Laufzeit: Wert an Index auslesen/ersetzen?
- ▶ Annahme: RAM Maschine
- ▶ Laufzeit: 6. Element anhängen?
- ▶ Element hinzufügen/löschen → neues Array mit entspr. Größe anlegen und kopieren

- ▶ words = string[3] {"A", "hallo", "123"}
- ▶ "feste Blockgröße"?
- ▶ Speicherbedarf?

Leetcode

- ▶ <https://leetcode.com/problems/majority-element/>
- ▶ Given an array `nums` of size n , return the majority element.
- ▶ The majority element is the element that appears more than $\lfloor n/2 \rfloor$ times. You may assume that the majority element always exists in the array.
- ▶ Direkt auf Leetcode lösen, alternativ eigene PythonEnv oder PseudoCode

Übung
Array Majority

Majority Element - Exhaustive Search

- ① Zähle und Speichere die Häufigkeit jedes Elements in ein neues int Array der Länge n
- ② Finde den Index mit dem größtem Wert in diesem Häufigkeitsarray
- ③ Gebe das Element aus `nums` mit diesem Index zurück

```
1 MajorityElement(int [] X):
2     counts = int[len(X)] //new array of len X with all 0s
3     for i in range(0,len(X)):
4         el = X[i]
5         foreach x in X:
6             if x == el:
7                 counts[i] += 1 //calc all counts
8     majEl = X[0]
9     majVal = counts[0]
10    for i in range(0,len(X)):
11        if counts[i] > majVal:
12            majEl = X[i]
13            majVal = counts[i] //calc max count
14    return majEl
```

```
1 MajorityElement(int [] X):
2     counts = int[len(X)] //n*4 bytes
3     for i in range(0,len(X)): //n steps
4         el = X[i]
5         foreach x in X: //n inner steps
6             if x == el:
7                 counts[i] += 1
8     majEl = X[0] //4 bytes
9     majVal = counts[0] //4 bytes
10    for i in range(0,len(X)): // n steps
11        if counts[i] > majVal:
12            majEl = X[i]
13            majVal = counts[i]
14    return majEl
```

```
1 MajorityElement(int [] X):
2     counts = int[len(X)] //n*4 bytes
3     for i in range(0,len(X)): //n steps
4         el = X[i]
5             foreach x in X: //n inner steps
6                 if x == el:
7                     counts[i] += 1
8             majEl = X[0] //4 bytes
9             majVal = counts[0] //4 bytes
10            for i in range(0,len(X)): // n steps
11                if counts[i] > majVal:
12                    majEl = X[i]
13                    majVal = counts[i]
14            return majEl
```

$8 + 4n$ Bytes; $n + n^2$ Steps

```
1 MajorityElement(int [] X):
2     counts = int[len(X)] //n*4 bytes
3     for i in range(0,len(X)): //n steps
4         el = X[i]
5             foreach x in X: //n inner steps
6                 if x == el:
7                     counts[i] += 1
8             majEl = X[0] //4 bytes
9             majVal = counts[0] //4 bytes
10            for i in range(0,len(X)): // n steps
11                if counts[i] > majVal:
12                    majEl = X[i]
13                    majVal = counts[i]
14    return majEl
```

$\mathcal{O}(n)$ Space; $\mathcal{O}(n^2)$ Time

- ▶ Optimierte Exhaustive Search
- ▶ HashMap / Dictionary
- ▶ Sortieren
- ▶ Randomisieren
- ▶ Divide and Conquer
- ▶ Boyer-Moore Voting³

³Robert S Boyer et al. “MJRTY—a fast majority vote algorithm”. In: **Automated Reasoning**. Springer, 1991, pp. 105–117.

Majority Element - Boyer-Moore Voting

- ① Setze einen int Zähler auf 0
- ② Nehme an, das erste Element ist das Mehrheitselement
- ③ Vergleiche das nächste Element mit dem aktuellen Kandidaten
- ④ Ist der Zähler auf 0, mache das aktuelle Element zum neuen Kandidaten
- ⑤ Ist das aktuelle Element gleich dem Kandidaten, addiere 1 auf den Zähler, sonst subtrahiere 1 vom Zähler
- ⑥ Wiederhole 3,4,5 für jedes Element im Array
- ⑦ Der aktuelle Kandidat ist das Mehrheitselement

```
1 MajorityElement(int [] X):  
2     count = 0 //4 bytes  
3     candidate = X[0] //4 bytes  
4     foreach x in X: //n steps  
5         if count == 0:  
6             candidate = x  
7             count += (1 if num == candidate else -1)  
8     return candidate
```

Bis zum nächsten Mal

Übungsaufgabe für Bonuspunkte

Eine der folgenden Leetcodes

- ▶ <https://leetcode.com/problems/longest-common-prefix/>
- ▶ <https://leetcode.com/problems/find-greatest-common-divisor-of-array/>
- ▶ Reminder: Submission Screenshot + Quellcode txt (mit Name / Matrik)
- ▶ Gruppenarbeit möglich (1-3 Studierende)
- ▶ Individuelle Moodle Abgabe

Literaturverzeichnis

Literaturverzeichnis I

-  Boyer, Robert S and J Strother Moore. "MJRTY—a fast majority vote algorithm". In: **Automated Reasoning**. Springer, 1991, pp. 105–117.
-  Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. **Introduction to Algorithms, Third Edition**. 3rd. The MIT Press, 2009. ISBN: 0262033844.
-  Skiena, Steven S. **The Algorithm Design Manual**. 2nd. Springer Publishing Company, Incorporated, 2008. ISBN: 1848000693.

Algorithmen und Datenstrukturen

Lecture 2

Prof. Dr. Maximilian Scherer
maximilian.scherer@dhw.de



SoSe 2025

Nachbesprechung

- ▶ Algorithmen
- ▶ Datenstrukturen
- ▶ Statische Arrays
- ▶ Majority Element
- ▶ Boyer-Moore Voting

Laufzeitbestimmung und O-Notation

```
1 l = [1,4,-6,0,4,8,4,7,-6,9]
2 #find maximum
3 m = l[0]
4 for x in l:
5   ^^Iif x > m:
6     ^^I^Im = x #update maximum
7 print(f"maximum is {m}")
```

- ▶ n Vergleiche
- ▶ 0 bis n Zuweisungen
- ▶ $n + n = 2n \in \mathcal{O}(n)$
- ▶ $m = \max(l)?$

- ▶ Annahme: Primitive Operationen kosten “1” / werden von Hardware + OS ausgeführt
- ▶ Arithmetik (Sonderfall Python int)
- ▶ Pointer Dereferenzieren

O-Notation

- ▶ Laufzeit- und Speicherbestimmung
- ▶ Asymptotische Schranke
- ▶ Menge aller Funktionen, die maximal wie eine gegebene Funktion g anwachsen
- ▶ $f \in \mathcal{O}(g) \leftrightarrow \limsup_{x \rightarrow a} \left| \frac{f(x)}{g(x)} \right| < \infty$
- ▶ Details in der Praxis oft wichtig, aber aus theoretischer Sicht nicht

O-Notation: Beispiele

- ▶ $5n \in \mathcal{O}(\quad)$
- ▶ $5n \in \mathcal{O}(n)$
- ▶ $34563456 \in \mathcal{O}(\quad)$
- ▶ $34563456 \in \mathcal{O}(1)$
- ▶ $23 + 17n + 0.002n^2 \in \mathcal{O}(\quad)$
- ▶ $23 + 17n + 0.002n^2 \in \mathcal{O}(n^2)$
- ▶ $2\log(5n) \in \mathcal{O}(\quad)$
- ▶ $2\log(5n) \in \mathcal{O}(\log n)$
- ▶ $n^{972342} + 2^n \in \mathcal{O}(\quad)$
- ▶ $n^{972342} + 2^n \in \mathcal{O}(2^n)$

O-Notation: Komplexitätsklassen (aufsteigend)

- ▶ $\mathcal{O}(1)$ konstant
- ▶ $\mathcal{O}(\log n)$ logarithmisch
- ▶ $\mathcal{O}(n)$ linear
- ▶ $\mathcal{O}(n \log n)$ super linear
- ▶ $\mathcal{O}(n^2)$ quadratisch
- ▶ $\mathcal{O}(n^m)$ polynomiell
- ▶ $\mathcal{O}(2^n)$ exponentiell
- ▶ $\mathcal{O}(n!) = \mathcal{O}(n^n)$ faktoriell / super exponentiell

- ▶ Beispiel für Algorithmus mit super exponentieller Laufzeit:
- ▶ Jede mögliche Reihenfolge eines 52-Kartendecks generieren.
- ▶ Wie viele Möglichkeiten?
- ▶ $52!$
- ▶ Wie lange dauert es, alle Möglichkeiten zu generieren?

<https://youtu.be/0biqJzfyACM?t=860>

Vorstellung Übungsaufgabe

- ▶ Kurzvorstellung Code
- ▶ Laufzeit / Speicher Analyse
 - <https://leetcode.com/problems/longest-common-prefix/>
 - <https://leetcode.com/problems/find-greatest-common-divisor-of-array/>

Greatest Common Divisors

```
1 def findGCD(self, nums: List[int]) -> int:
2     a = min(nums)
3     b = max(nums)
4     for i in range(a, 1, -1):
5         if a % i == 0 and b % i == 0:
6             return i
```

$\mathcal{O}(\min(a, b) + n)$ linear time, $\mathcal{O}(1)$ constant space

```
1 def findGCD(self, nums: List[int]) -> int:
2     # get min and max values of array (linear runtime)
3     a = min(nums)
4     b = max(nums)
5     while a != b:
6         if b > a:
7             b -= a
8         else:
9             a -= b
10    return a
```

$\mathcal{O}(\max(a, b) + n)$ linear time, $\mathcal{O}(1)$ constant space

```
1 def findGCD(self, nums: List[int]) -> int:
2     a = min(nums)
3     b = max(nums)
4     r = b % a
5     while r != 0:
6         b = a
7         a = r
8         r = b % a
9
10    return a
```

$\mathcal{O}(\log(\max(a, b)) + n)$ time, $\mathcal{O}(1)$ space

Longest Common Prefix

```
1 def longestCommonPrefix(self, strs: List[str]) -> str:
2     #assume first word is prefix
3     pre = strs[0]
4
5     for s in strs:
6         while not s.startswith(pre):
7             #shorten prefix
8             pre = pre[:-1]
9
10    return pre
```

$\mathcal{O}(n)$ time ($n = \text{num characters}$), $\mathcal{O}(1)$ space

```
1 def longestCommonPrefix(self, strs: List[str]) -> str:
2     if not strs or not strs[0]:
3         return "" #edge cases
4
5     #grow prefix
6     i = 0
7     while True:
8         pre = strs[0][0:(i+1)]
9         for s in strs:
10             if len(s) <= i:
11                 return s
12             if s[i] != pre[-1]:
13                 return pre[:-1]
14
15     i += 1
```

$\mathcal{O}(n)$ time ($n = \text{num characters}$), $\mathcal{O}(1)$ space

Python Memory Management

- ▶ Python manages Speicher automatisch (Garbage Collector)
- ▶ ALLES in Python sind Objekte
- ▶ Ja ALLES, auch primitive wie bool,int, float
- ▶ Zuweisungen kopieren Daten nur wenn nötig
- ▶ <https://www.youtube.com/watch?v=Bz3ir-vKqkk>
- ▶ <https://pythontutor.com/visualize.html>

Datenstruktur: Dynamisches Array

- ▶ Was ist ein dynamisches Array? Eine “Liste”?
- ▶ dynamischer, zusammenhängender Speicherbereich fester Blockgröße
- ▶ Operationen
 - Hinzufügen
 - Löschen
- ▶ Möglichst schnell

- ▶ <https://www.youtube.com/watch?v=qTb1sZX74K0>
- ▶ https://www.youtube.com/watch?v=5AllG-i_yto
- ▶ <https://www.youtube.com/watch?v=Ij7NQ-0mIVA>

Dynamisches Array

- ▶ Statisches Array im Hintergrund
- ▶ Zähler wie viele Elemente vorhanden sind
- ▶ Bei Bedarf Größe verdoppeln (teuer aber selten)
- ▶ Amortisierte Laufzeit bei Anhängen:
 - n Elemente anhängen je $\mathcal{O}(1)$
 - Nächstes Element anhängen $\mathcal{O}(n)$ (alle Elemente kopieren)
 - Also $2n$ Rechenoperation um $n + 1$ Elemente einzufügen
 - Kosten: $2n/(n + 1) \in \mathcal{O}(1)$
- ▶ Praxis: Vermutlich benötigte Größe angeben

- ▶ Gegeben ein Array X und Zähler N für die Anzahl Elemente im Array
- ▶ Schreiben Sie PseudoCode, um ein Element an Index $i \in [0, N - 1]$ zu löschen
- ▶ Analysieren Sie die Laufzeit
- ▶ Ergänzung: Kann Ihr Code beschleunigt werden, wenn die Reihenfolge der Elemente in X egal ist?

Übung

Array Element removal

- ▶ Was ist schneller, ein Element vorne oder hinten löschen? Warum?
- ▶ Wenn die Reihenfolge der Elemente egal ist, wie kann dann das Löschen beschleunigt werden?
- ▶ Swap Delete:
 - Element an Stelle i mit letztem Element tauschen
 - letztes Element löschen
 - Speicher und Laufzeit $\mathcal{O}(1)$
- ▶ Python Listen “[]” sind als dynamische Arrays implementiert¹

¹<http://www.laurentluce.com/posts/python-list-implementation/>

Mehrdimensionale Arrays

- ▶ Array ist lineare Sequenz / Kette von Daten.
- ▶ Wie wird z.B. ein Schwarz-Weiß-Bild (2D Matrix) als Array gespeichert?
- ▶ Als Sequenz aller Pixel in Spalten- oder Zeilenreihenfolge
- ▶ Zugriff über 2D Index möglich

Übung

<https://leetcode.com/problems/flood-fill/>

Bis zum nächsten Mal

Übungsaufgabe für Bonuspunkte

Eine der folgenden Leetcodes

- ▶ <https://leetcode.com/problems/remove-element/>
- ▶ <https://leetcode.com/problems/two-sum/>
- ▶ <https://leetcode.com/problems/search-a-2d-matrix/>
- ▶ Reminder: Submission Screenshot + Quellcode txt (mit Name / Matrik)
- ▶ Gruppenabgabe möglich (1-3 Studierende)

Algorithmen und Datenstrukturen

Lecture 3

Prof. Dr. Maximilian Scherer
maximilian.scherer@dhw.de



SoSe 2025

Nachbesprechung

- ▶ O-Notation
- ▶ Laufzeit und Speicherbedarf
- ▶ Dynamische Arrays
- ▶ Einfügen, Entfernen

Vorstellung Übungsaufgabe

- ▶ Kurzvorstellung Code
- ▶ Laufzeit / Speicher Analyse
 - <https://leetcode.com/problems/remove-element/>
 - <https://leetcode.com/problems/two-sum/>
 - <https://leetcode.com/problems/search-a-2d-matrix/>

```
1 def removeElement(self, nums: List[int], val: int) -> int:
2     to_del = []
3     for i,e in enumerate(nums):
4         if e==val:
5             to_del.append(i)
6     print(to_del)
7     for i in to_del:
8         List.pop(i)
9
10    return len(to_del)
```

$\mathcal{O}(n^2)$ time, $\mathcal{O}(n)$ space

```
1 def removeElement(self, nums: List[int], val: int) -> int:
2     to_del = []
3     for i,e in enumerate(nums):
4         if e==val:
5             to_del.append(i)
6     print(to_del)
7     for i in to_del:
8         List[i],List[-1] = List[-1],List[i] #swap
9         List.pop() #remove last
10
11    return len(to_del)
```

$\mathcal{O}(n)$ time, $\mathcal{O}(n)$ space

```
1 def removeElement(self, nums: List[int], val: int) -> int:
2     while val in nums:
3         nums.remove(val)
4     return len(nums)
```

$\mathcal{O}(n^2)$ time, $\mathcal{O}(1)$ space

```
1 def removeElement(self, nums: List[int], val: int) -> int:
2     i = 0
3     for x in nums:
4         if x != val:
5             nums[i] = x
6             i += 1
7     return i
```

$\mathcal{O}(n)$ time, $\mathcal{O}(1)$ space

```
1 def twoSum(self, nums: List[int], target: int) -> List[int]:  
2     for i,x in enumerate(nums):  
3         for j,y in enumerate(nums):  
4             if i==j:  
5                 continue  
6             if x+y==target:  
7                 return i,j
```

$\mathcal{O}(n^2)$ time, $\mathcal{O}(1)$ space

```
1 def twoSum(self, nums: List[int], target: int) -> List[int]:  
2     seen = {}  
3     for i,e in enumerate(nums):  
4         complement = target - e  
5         if complement in seen:  
6             return [seen[complement], i]  
7         else:  
8             seen[e] = i
```

$\mathcal{O}(n)$ time, $\mathcal{O}(n)$ space

```
1 def searchMatrix(self, matrix: List[List[int]], target: int):
2     for row in matrix:
3         if target in row:
4             return True
5     return False
```

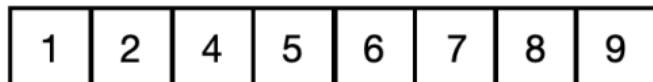
$\mathcal{O}(n \cdot m)$ time, $\mathcal{O}(1)$ space

```
1 def searchMatrix(self, matrix: List[List[int]], target: int):
2     for row in matrix:
3         if row[-1] >= target:
4             return target in row
5     return False
```

$\mathcal{O}(n \cdot m)$ time, $\mathcal{O}(1)$ space

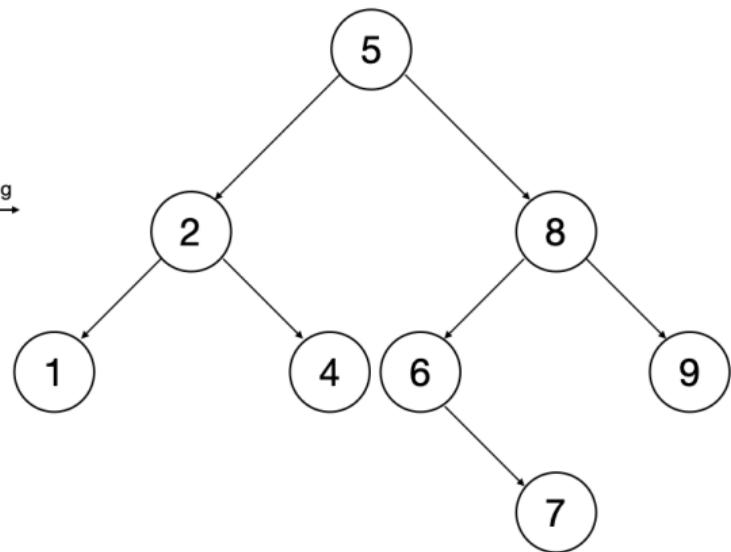
- ▶ Gleiches Problem: Pin Code eines Schlosses knacken
- ▶ Orakel: Ist der eingegebene Code größer oder kleiner als die Lösung
- ▶ Algorithmus der schneller ist, als alle Kombinationen probieren?
- ▶ Binärsuche

Sorted Array



Converting

Balanced binary tree



```
1 def searchSortedList(lsorted: List[int], target: int):
2     while True:
3         if not lsorted: return False
4         mid = len(lsorted) // 2
5         if lsorted[mid] == target:
6             return True
7         if lsorted[mid] < target:
8             lsorted = lsorted[(mid+1):]
9         else:
10            lsorted = lsorted[:mid]
```

$\mathcal{O}(\log(n))$ time, $\mathcal{O}(1)$ space

```
1 def searchMatrix(self, matrix: List[List[int]], target: int):
2     nrow = len(matrix)
3     ncol = len(matrix[0])
4     i_s = 0
5     i_e = nrow*ncol
6     while i_s < i_e:
7         mid = i_s + (i_e - i_s)//2
8         e_mid = matrix[mid//ncol][mid%ncol] #idx2coord
9         if e_mid==target:
10             return True
11         if e_mid < target:
12             i_s = mid+1
13         else:
14             i_e = mid
15     return False
```

$\mathcal{O}(\log(n \cdot m))$ time, $\mathcal{O}(1)$ space

Übung

<https://leetcode.com/problems/flood-fill/>

Recursive Flood Fill

```
1 target_color = image[sr][sc]
2 rows,cols = len(image),len(image[0])
3 # Recursive function to perform flood fill
4 def paint_fill(i,j):
5     # index is out of bounds
6     if i<0 or i>=rows or j<0 or j>=cols: return
7     # current pixel is not target color or already replaced
8     if image[i][j] != target_color or image[i][j]==color:
9         return
10    # Set the color of the current pixel to the new color
11    grid[i][j] = color
12    # Recursively flood fill on the 4-directional neighbors
13    paint_fill(i-1,j) # up
14    paint_fill(i,j-1) # left
15    paint_fill(i+1,j) # down
16    paint_fill(i,j+1) # right
17 # start recursive filling from starting pixel
18 paint_fill(sr,sc)
```

$\mathcal{O}(n \cdot m)$ time, $\mathcal{O}(n \cdot m)$ space

Iterative Flood Fill

```
1 target_color = image[sr][sc]
2 rows,cols = len(image),len(image[0])
3 # dynamic array (used as stack)
4 neighbors = [(sr,sc)]
5 while neighbors:
6     #get next neighbor
7     i,j = neighbors.pop()
8     # index is out of bounds
9     if i<0 or i>=rows or j<0 or j>=cols: continue
10    # current pixel is not target color or already replaced
11    if image[i][j] != target_color or image[i][j]==color:
12        continue
13    # Set the color of the current pixel to the new color
14    image[i][j] = color
15    # append 4-directional neighbors of the current pixel
16    neighbors.append((i-1,j)) # up
17    neighbors.append((i,j-1)) # left
18    neighbors.append((i+1,j)) # down
19    neighbors.append((i,j+1)) # right
```

$\mathcal{O}(n \cdot m)$ time, $\mathcal{O}(n \cdot m)$ space

Datenstruktur: Verkettete Listen

Verkettete Liste / Linked List

- ▶ Alternative Datenstruktur zum dynamischen Array
- ▶ Kein zusammenhängender Speicherbereich
- ▶ Eine verkette Liste besteht aus Knoten
- ▶ Jeder Knoten besitzt:
 - Einen Wert (integer, float, string, ...)
 - Einen Zeiger auf den nachfolgenden Knoten in der Liste
 - Ggf. Zeiger auf den Vorgänger-Knoten (double linked list)

Übung

Implementieren Sie eine Verkettete Liste als Klasse in Python (ohne Methoden)

```
1 class ListNode:  
2     def __init__(self, x):  
3         self.val = x #value of this node  
4         self.next = None #pointer to the next node  
5     #example usage to create list [1,2,5]  
6     myList = ListNode(1)  
7     myList.next = ListNode(2)  
8     myList.next.next = ListNode(5)
```

Operationen auf Linked Lists

- ▶ Zugriff auf Element i
- ▶ Hinzufügen (irgendwo)
- ▶ Löschen (irgendwo)

Get

```
1 //retrieve Node at index
2 Get(Node head, int index): //get node at index i
3     node = head //given head Node
4     for j in range(index-1):
5         node = node.next
6     return node
```

Get

```
1 //retrieve Node i
2 Get(Node head, int index):
3     node = head //1 space
4     for j in range(index-1): //n steps
5         node = node.next
6     return node
```

- ▶ Get: $\mathcal{O}(n)$ time und $\mathcal{O}(1)$ space
- ▶ Vergleich zu Array?

Insert

```
1 Insert(Node before, Node next):  
2     //insert node after before  
3     dummy = before.next //temp save next  
4     before.next = next //insert element  
5     next.next = dummy //restore connection to rest of ←  
       list
```

Insert

```
1 Insert(Node before, Node next):  
2     //insert node after before  
3     dummy = before.next //1 space, 1 step  
4     before.next = next //1 step  
5     next.next = dummy //1 step
```

Insert is $\mathcal{O}(1)$ time and $\mathcal{O}(1)$ space

Delete

Übung

<https://leetcode.com/problems/delete-node-in-a-linked-list>

```
1 def deleteNode(self, node):
2     #replace node with next node:
3     #move value from next node into current node
4     node.val = node.next.val
5     #point current node to next next node
6     node.next = node.next.next
```

Linked List Zusammenfassung

- ▶ Zeiger-basierte Datenstruktur (vs. Zusammenhängender Speicher)
- ▶ Schnelles einfügen und löschen: $\mathcal{O}(1)$
- ▶ Langsames Zugreifen: $\mathcal{O}(n)$
- ▶ Overhead: Element + Zeiger (bei Integer Elementen also 50% Overhead!)
- ▶ Steven Skiena Lecture 4:

<https://www.youtube.com/watch?v=vg2u8Hbb6lE&t=2592s>

- ▶ Erweiterung: Doubly Linked List (prev und next pointer)

Übungsaufgabe für Bonuspunkte

Leetcode

- ▶ <https://leetcode.com/problems/middle-of-the-linked-list/>
- ▶ Reminder: Submission Screenshot + Quellcode txt (mit Name / Matrik)
- ▶ Gruppenabgabe möglich (1-3 Studierende)

Algorithmen und Datenstrukturen

Lecture 4

Prof. Dr. Maximilian Scherer
maximilian.scherer@dhw.de



SoSe 2025

Nachbesprechung

- ▶ Verkette Listen
- ▶ Operationen auf LinkedList und Vergleich mit Array

Vorstellung Übungsaufgabe

- ▶ Kurzvorstellung Code
- ▶ Laufzeit / Speicher Analyse

Verlinkte Liste in Array übertragen; Mittlere Position ausgeben

```
1 def middleNode(self, head: ListNode) -> ListNode:
2     node = head
3     c = 0
4     while node:
5         c += 1
6         node = node.next
7     node = head
8     for i in range(c//2):
9         node = node.next
10    return node
```

$\mathcal{O}(n)$ time, $\mathcal{O}(1)$ space

```
1 def middleNode(self, head: ListNode) -> ListNode:
2     arr = [head]
3     while arr[-1].next:
4         arr.append(arr[-1].next)
5     return arr[len(arr) // 2]
```

$\mathcal{O}(n)$ time, $\mathcal{O}(n)$ space

Langsamer und schneller Pointer.

```
1 def middleNode(self, head: ListNode) -> ListNode:
2     slow = fast = head
3     while fast and fast.next:
4         slow = slow.next
5         fast = fast.next.next
6     return slow
```

$\mathcal{O}(n)$ time, $\mathcal{O}(1)$ space middle linked list.mp4

HashMap

set

- ▶ set entspricht der mathematischen Definition einer Menge
- ▶ ungeordnet
- ▶ keine Duplikate
- ▶ $\mathcal{O}(n)$ zum Suchen eines Elements

- ▶ Interne Funktionsweise?
- ▶ Hashfunktion (Speicheradresse oder inhaltsbasiert)
- ▶ Modulo
- ▶ Chaining mit Linked List
- ▶ <https://www.youtube.com/watch?v=shs0KM3wKv8>

Sortieren

- ▶ Was bedeutet sortieren?
- ▶ Ordinale Daten / Vergleichbare Daten
- ▶ in auf- oder absteigende Reihenfolge bringen
- ▶ Zur Veranschaulichung nutzen wir meist ganze Zahlen und sortieren aufsteigend

```
1 l = [4,6,3,57,90,-7,7,99,3]  
2 l.sort() # done :)
```

- ▶ Funktionsweise?
- ▶ Laufzeit? Speicherbedarf?
- ▶ Worst-case? Best-case?

Übung

- ▶ Sortieren Sie n Integer-Zahlen aufsteigend
- ▶ Versuchen Sie Ihren Algorithmus in Worte / Pseudocode / Pythoncode zu fassen
- ▶ Was ist die Laufzeit / Speicherbedarf?
- ▶ Beispieldaten: 9, 8, 5, 0, 4, 2

- ▶ Bubblesort
- ▶ Selection sort
- ▶ Insertion sort
- ▶ Heapsort
- ▶ Mergesort
- ▶ Quicksort
- ▶ Bucketsort
- ▶ Bogosort / Sleepsort / Miraclesort

- ▶ $\mathcal{O}(n^2)$
 - Bubblesort
 - Selection sort
 - Insertion sort
- ▶ $\mathcal{O}(n \log n)$
 - Heapsort
 - Mergesort
 - Quicksort
- ▶ Bucket sort $\mathcal{O}(n)$

- ▶ $\mathcal{O}(n^2)$
 - Bubblesort
 - Selection sort
 - Insertion sort
- ▶ $\mathcal{O}(n \log n)$
 - Heapsort
 - Mergesort
 - Quicksort
- ▶ Bucket sort $\mathcal{O}(n)$

- ▶ Sortierung basiert (meist) auf paarweisen Vergleichen
- ▶ Vergleichsoperatoren für numerische Werte klar
- ▶ Andere Datentypen / Klassen benötigen definierten Vergleichsoperator

```
1 class Point:  
2     def __init__(self, x,y):  
3         self.x=float(x)  
4         self.y=float(y)
```

```
1 class Point:  
2     ^~Idef __init__(self, x,y):  
3         ^~Iself.x:float = x  
4         ^~Iself.y:float = y  
5     ^~Idef __lt__(self, other):  
6         ^~I...  
7
```

- ▶ Visualisierung und Vertonung verschiedener Sortieralgorithmen
- ▶ <https://www.youtube.com/watch?v=kPRA0W1kECg>

BubbleSort

- ▶ Einfach und intuitiv
- ▶ bubblesort.gif

BubbleSort – In Worten

- ① Wähle jedes Element i
- ② Wähle jedes Elemente j bis auf die letzten i Stück
- ③ Wenn Element j größer als sein rechter Nachbar ($j+1$) ist, vertausche beide

```
1 def bubble_sort(array):
2     n = len(array)
3     for i in range(n):
4         # last i elements will be sorted
5         for j in range(n - i - 1):
6             #compare two adjancent values and swap if required
7             if array[j] > array[j + 1]:
8                 array[j], array[j + 1] = array[j + 1], array[j]
9
10    return array
```

$\mathcal{O}(n^2)$ time, $\mathcal{O}(1)$ space

Insertion sort

- ▶ Einfach und intuitiv
- ▶ Der Schnellste unter den “langsamem” Sortieralgorithmen
- ▶ Gleiche Performance auf Arrays und verketteten Listen
- ▶ Insertionsort.gif

Insertion sort – In Worten

- ① Wähle das nächste Element
- ② Betrachte alle Elemente links des gewählten Elements
(von rechts nach links)
 - Wenn das Element größer ist, bewege es nach rechts
- ③ wiederhole 1 und 2 für alle Elemente

Insertion sort – Code

```
1 def insertion_sort(array):
2     for i in range(1, len(array)): #iterate array
3         key_item = array[i] #element to be sorted
4         j = i - 1
5         while j >= 0 and array[j] > key_item:
6             #shift all larger elements to the right
7             array[j + 1] = array[j]
8             j -= 1
9         array[j + 1] = key_item #insert at correct pos
```

Insertion sort – Analysis

```
1 def insertion_sort(array):
2     for i in range(1, len(array)): #n steps
3         key_item = array[i]
4         j = i - 1
5         while j >= 0 and array[j] > key_item:
6             #n inner steps
7             array[j + 1] = array[j]
8             j -= 1
9         array[j + 1] = key_item
```

$\mathcal{O}(n^2)$ time, $\mathcal{O}(1)$ space

Mergesort

- ▶ Schneller Sortieralgorithmus
- ▶ Iterative und Rekursive Formulierung
- ▶ Teile und Herrsche Prinzip (divide and conquer)
- ▶ mergesort.gif

Mergesort – Divide and Conquer

- ① Teile alle Elemente in zwei Gruppen
- ② Wiederhole 1 bis nur noch ein Element in jeder Gruppe ist
- ③ Füge die Gruppen vergleichsbasiert zusammen

```
1 def merge_sort(array):
2     if len(array) <= 1:
3         return array #single element is already sorted
4
5     midIdx = len(array)//2
6     left = merge_sort(array[:midIdx]) #recurse into left
7     right = merge_sort(array[midIdx:]) #recurse into right
8     return merge(left,right) #merge sorted sublists
9
10 def merge(left,right):
11     result = []
12     i = 0 #left idx counter
13     j = 0 #right idx counter
14     while i < len(left) and j < len(right):
15         if left[i] <= right[j]:
16             result.append(left[i]) #interleave left
17             i += 1 #next left element
18         else:
19             result.append(right[j]) #interleave right
20             j += 1 #next right element
21     return result + left[i:] + right[j:] #append remaining elements
```

```

1 def merge_sort(array):
2     if len(array) <= 1:
3         return array
4
5     midIdx = len(array)//2
6     left = merge_sort(array[:midIdx]) #log n steps
7     right = merge_sort(array[midIdx:]) #log n steps
8     return merge(left,right) #log n steps
9
10 def merge(left,right):
11     result = []
12     i = 0
13     j = 0
14     while i < len(left) and j < len(right): #n steps
15         if left[i] <= right[j]:
16             result.append(left[i]) #n space
17             i += 1
18         else:
19             result.append(right[j]) #n space
20             j += 1
21     return result + left[i:] + right[j:] #n steps

```

$\mathcal{O}(n \log n)$ time, $\mathcal{O}(n)$ space

Bucket Sort / Bin Sort

- ▶ Lineare Laufzeit unter gewissen Annahmen
- ▶ Idee:
 - Elemente nach Größe in Eimer eingruppieren
 - Jeden Eimer sortieren
 - Elemente der Eimer konkatenieren
- ▶ Annahme: Uniforme Verteilung der Daten

```
1 def bucket_sort(array, k):
2     buckets = []
3     for i in range(k): #k << n steps
4         buckets.append([])
5     min_el = min(array)
6     max_el = float(max(array)-min_el)+1
7     for i in range(len(array)): #n steps
8         el = array[i]
9         buckets[int((el-min_el)/max_el * k)].append(el) #n space
10    outlist = []
11    for i in range(k): #k << n steps
12        insertion_sort(buckets[i])
13        outlist.extend(buckets[i]) #n space
14    return outlist
```

$\mathcal{O}(n)$ time if uniform, $\mathcal{O}(n)$ space

Übung

<https://leetcode.com/problems/sort-an-array>

Live-Coding

Sorting in Python

Sorting Comparison

- ▶ <https://www.youtube.com/watch?v=qk7b4-iyCJ4>

Bis zum nächsten Mal

Übungsaufgabe für Bonuspunkte

Eine der folgenden Leetcodes

- ▶ <https://leetcode.com/problems/sort-list/>
- ▶ <https://leetcode.com/problems/sort-colors/>
- ▶ Reminder: Submission Screenshot + Quellcode txt (mit Name / Matrik)
- ▶ Gruppenabgabe möglich (1-3 Studierende)

Algorithmen und Datenstrukturen

Lecture 5

Prof. Dr. Maximilian Scherer
maximilian.scherer@dhw.de



SoSe 2025

Nachbesprechung

- ▶ Verkette Listen
- ▶ Operationen auf LinkedList und Vergleich mit Array
- ▶ Sortieren
- ▶ Bubble Sort / Insertion Sort
- ▶ Mergesort
- ▶ Bucketsort
- ▶ Sorting in Python

Vorstellung Übungsaufgabe

- ▶ Kurzvorstellung Übungsaufgabe
- ▶ Laufzeit / Speicher Analyse
- ▶ <https://leetcode.com/problems/sort-list/>
- ▶ <https://leetcode.com/problems/sort-colors/>

Sort Linked List: Idea

- ① Return empty or 1-element lists
- ② fill array with LL node values
- ③ sort array
- ④ build new LL with sorted values

```
1 def sortList(self, head) -> ListNode:
2     if not head or not head.next:
3         return head #nothing to do
4
5     arr = [] #convert to array
6     while head:
7         arr.append(head.val)
8         head = head.next
9     arr.sort() #sort array
10    newHead = tail = ListNode() #convert to LL
11    for i,e in enumerate(arr):
12        tail.val = e
13        if i < len(arr)-1:
14            tail.next = ListNode()
15            tail = tail.next
16    return newHead
```

$\mathcal{O}(n \log n)$ time, $\mathcal{O}(n)$ space

```
1 def sortList(self, head) -> ListNode:
2     if not head or not head.next:
3         return head #end recursion
4     left = head # Split the list into two halves
5     right = self.getMid(head) #see before
6     tmp = right.next
7     right.next = None #cutoff linked list in left
8     right = tmp
9     left = self.sortList(left)
10    right = self.sortList(right)
11    return self.merge(left, right)
12
13 def merge(self, list1, list2):
14     newHead = tail = ListNode() #save head and tail
15     while list1 and list2:
16         if list1.val > list2.val:
17             tail.next = list2; list2 = list2.next
18         else:
19             tail.next = list1; list1 = list1.next
20         tail = tail.next
21     if list1:
22         tail.next = list1
23     if list2:
24         tail.next = list2
25     return newHead.next # n logn time, n space
```

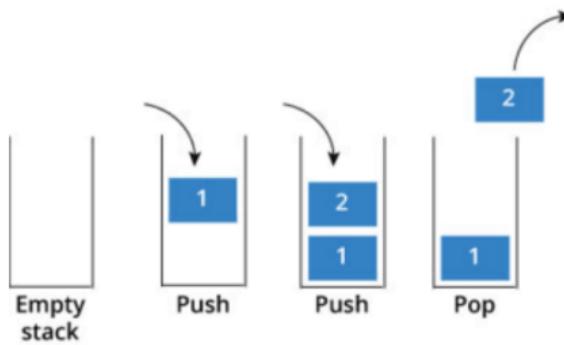
Sort Categorical Data In-Place

- ① Count how often each category occurs
- ② replace entries in nums according to category counts
- ③ profit

```
1 def sortColors(self, nums: List[int]) -> None:
2     r = 0
3     w = 0
4     b = 0
5     for i,e in enumerate(nums):
6         if e==0:
7             r += 1
8         elif e==1:
9             w += 1
10        else:
11            b += 1
12        for i in range(r):
13            nums[i] = 0
14        for i in range(r,r+w):
15            nums[i] = 1
16        for i in range(r+w,r+w+b):
17            nums[i] = 2
```

$\mathcal{O}(n)$ time, $\mathcal{O}(1)$ space

Datenstruktur: Stack und Queue



Stack



Queue

- ▶ Stack / Stapel
- ▶ LIFO - Last-in First-out
- ▶ Einsatz vs normales Array:
 - Reihenfolge umkehren
 - Priorisieren
 - Rekursion vermeiden
 - Tiefensuche
- ▶ In Python: Liste kann direkt als Stack verwendet werden. Warum?

```
1 stack = []
2 stack.append(1)
3 stack.append(2)
4 stack.append(3)
5 while stack:
6     print(stack.pop())
```

- ▶ Hinzufügen / Entfernen in $\mathcal{O}(1)$ Laufzeit
- ▶ Achtung: $\text{pop}(n)$ für $n \neq N-1$ ist $\mathcal{O}(n)$ Laufzeit

- ▶ Queue / Schlange
- ▶ FIFO - First-in First-out
- ▶ Einsatz vs normales Array:
 - Priorisieren
 - Verteilen / Parallelisieren (producer consumer queue)
 - Rekursion vermeiden
 - Breitensuche
- ▶ In Python: deque (als doubly linked list implementiert)

```
1 from collections import deque  
2 q = deque()  
3 q.append(1)  
4 q.append(2)  
5 q.append(3)  
6 while q:  
7     print(q.popleft())
```

- ▶ Hinzufügen / Entfernen in $\mathcal{O}(1)$ Laufzeit
- ▶ Auch als stack: `q.pop()` in $\mathcal{O}(1)$ Laufzeit

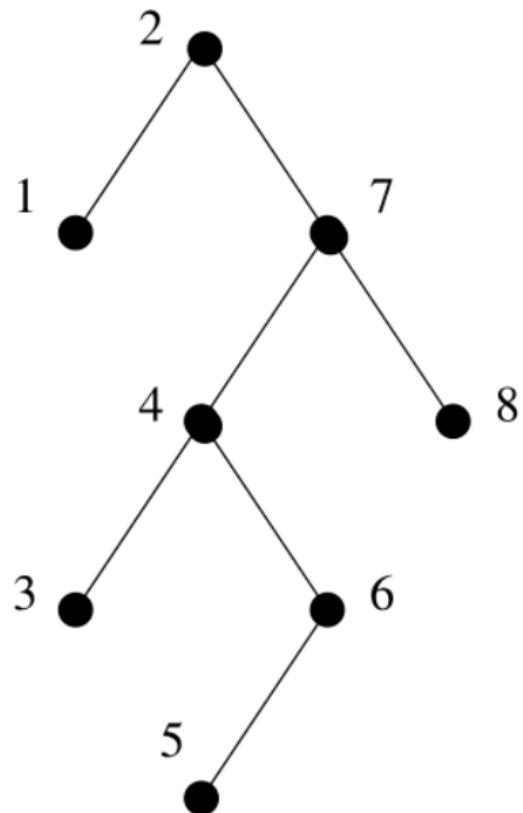
Datenstruktur: Bäume

- ▶ Datenstruktur für hierarchische Daten
 - Stammbaum
 - Dateisystem
 - Organisationsstruktur
- ▶ Schnelles Abändern
- ▶ Schnelles Suchen

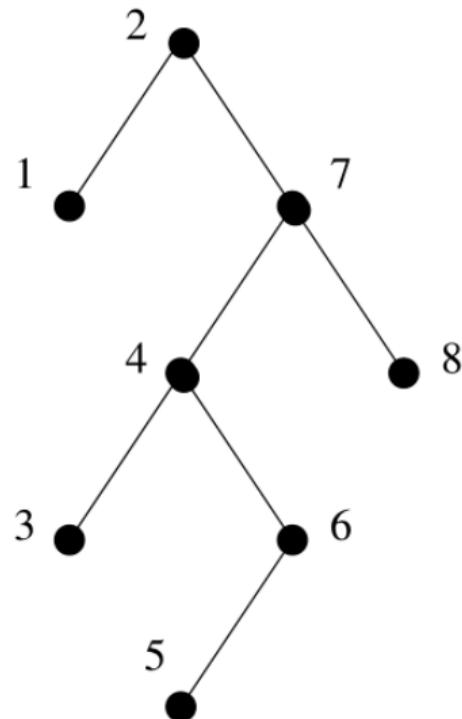
Baum?



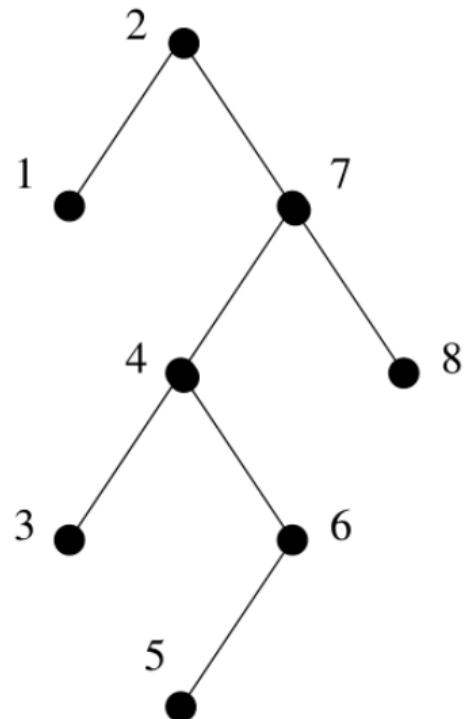
Baum!



- ▶ Definition T : Nicht-leere Menge von Knoten v und Kanten e , so dass genau ein Pfad zwischen zwei Knoten existiert
- ▶ Parent / Elternkonten
- ▶ Child / Kindknoten
- ▶ Root / Wurzel
- ▶ Sibling / Geschwister
- ▶ Interner Knoten
- ▶ Leaf / Blatt



- ▶ Stump / Stumpf / Kurzer Baum
- ▶ Subtree / Teilbaum
- ▶ Ordered Trees / Geordnete Bäume
- ▶ Binary Tree / Binärbaum
- ▶ Binary Search Tree / BST



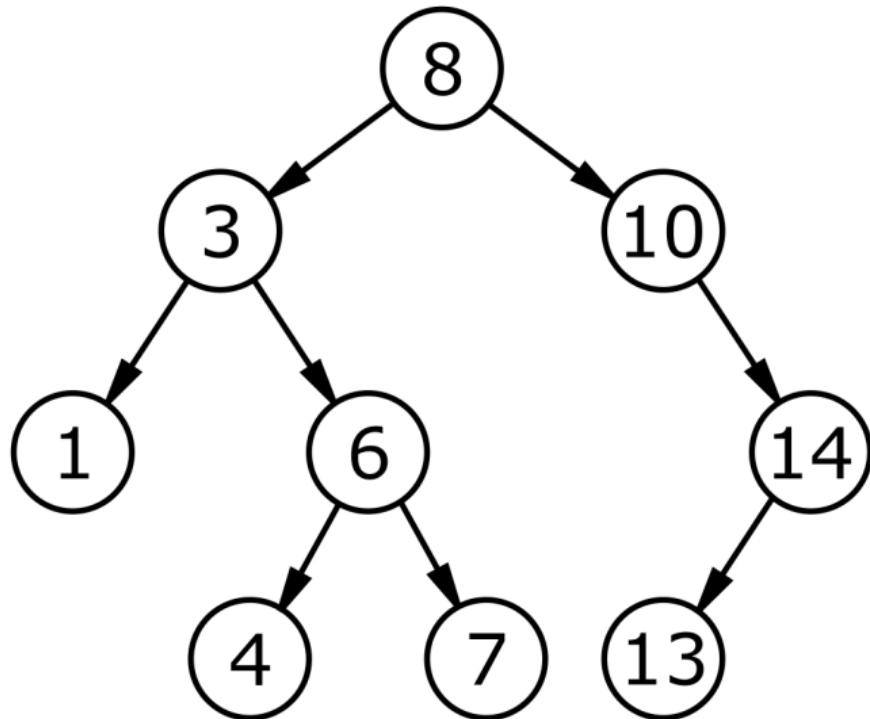
Übung

Implementieren Sie eine “Knoten”-Klasse für einen beliebigen Baum in Python

Operationen auf Bäumen

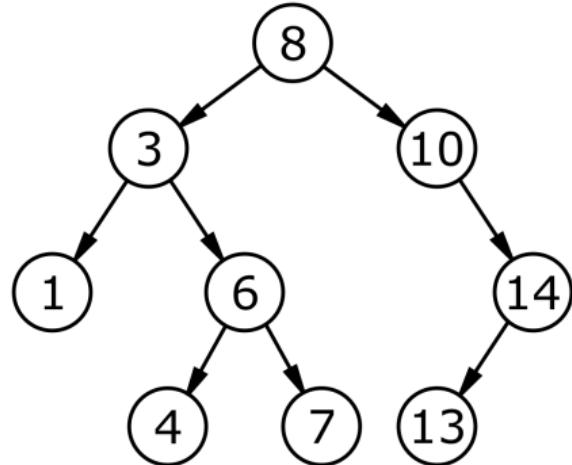
- ▶ Einfügen
- ▶ Löschen
- ▶ Traversieren
- ▶ Balancieren

Binary Search Tree



► Gesucht Element mit gegebenem Wert (z.B. die 7)

- ① Starte bei der Wurzel.
- ② Ist der aktuelle Knoten das gesuchte Element, gib es zurück und brich ab.
- ③ Ist der gesuchte Wert kleiner als der aktuelle Knoten, gehe zum linken Kindknoten, sonst zum rechten Kindknoten .
- ④ Wiederhole 2 und 3. Brich ab wenn ein Blatt erreicht wurde.



► Laufzeit $\log n$, wenn balanciert

Tree Traversal

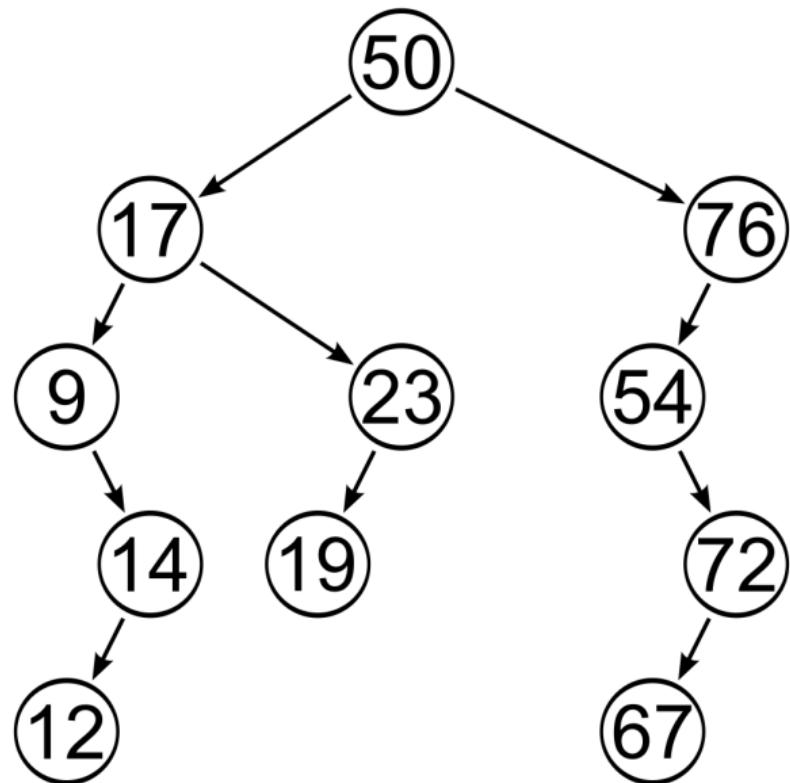
[https:](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/BinaryTreeTraversal.html)

//opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/BinaryTreeTraversal.html

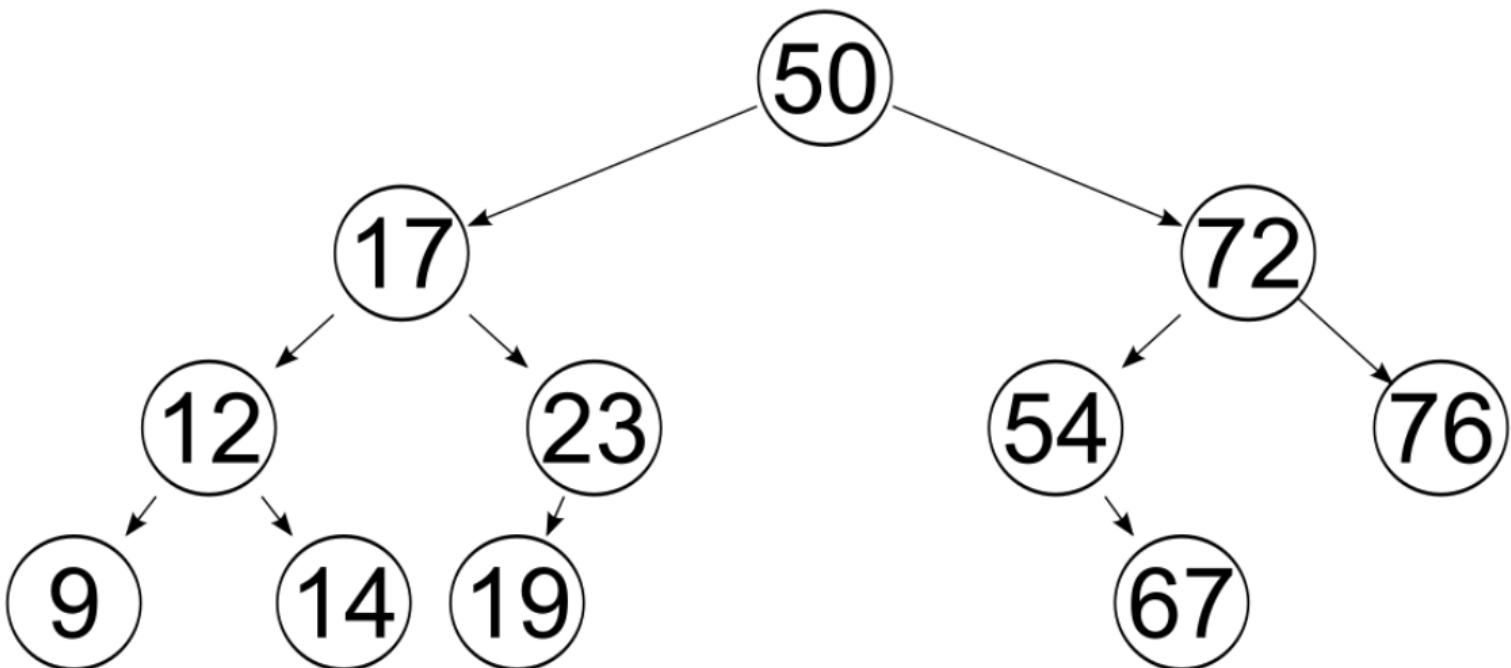
Binary Search Tree

- ▶ Einfügen
 - Wie bei verketteter Liste (Pointer “umbiegen”)
 - Einfügen: Nächst kleineres / größeres Element Suchen
- ▶ Löschen analog
- ▶ ggf. Rebalancieren nach einfügen / löschen

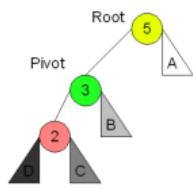
Self-balancing BST



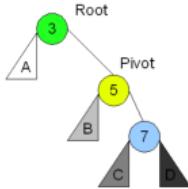
Self-balancing BST



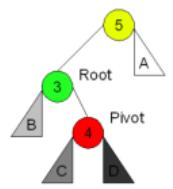
Left Left Case



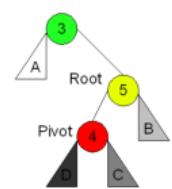
Right Right Case



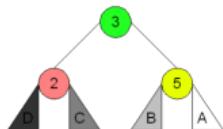
Left Right Case



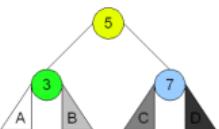
Right Left Case



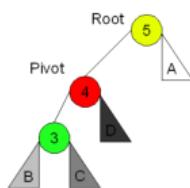
Right Rotation



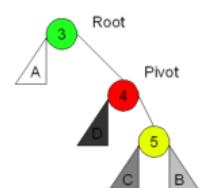
Left Rotation



Left Rotation



Right Rotation



AVL Tree / Self-balancing BST

- ▶ https://www.youtube.com/watch?v=-9sHvAnLN_w
- ▶ <https://visualgo.net/en/bst>

Live-Coding

BST Suche in Python

Übungsaufgabe für Bonuspunkte

Eine der folgenden Leetcodes

- ▶ <https://leetcode.com/problems/insert-into-a-binary-search-tree/>
- ▶ <https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/>
- ▶ Reminder: Submission Screenshot + Quellcode txt (mit Name / Matrik)

Algorithmen und Datenstrukturen

Lecture 6

Prof. Dr. Maximilian Scherer
maximilian.scherer@dhw.de



SoSe 2025

Nachbesprechung

- ▶ Bäume
- ▶ Operationen auf Bäumen
- ▶ Binary Search Tree
- ▶ AVL-Tress: Self-balancing BST

Vorstellung Übungsaufgabe

- ▶ Kurzvorstellung Code
- ▶ Laufzeit / Speicher Analyse
- ▶ <https://leetcode.com/problems/insert-into-a-binary-search-tree/>
- ▶ <https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/>

Insert into BST

- ① Return new Node with val if root is empty
- ② binary search
 - go right if val is greater than node
 - go left if val is less/equal than node
- ③ recurse until leaf

```
1 def insertIntoBST(self, root, val) -> Optional[TreeNode]:  
2     if root is None:  
3         return TreeNode(val)  
4     if root.val > val:  
5         root.left = self.insertIntoBST(root.left, val)  
6     else:  
7         root.right = self.insertIntoBST(root.right, val)  
8     return root
```

$\mathcal{O}(\log n)$ time, $\mathcal{O}(\log n)$ space

Insert into BST Iterative

- ① Return new Node with val if root is empty
- ② binary search
 - go right if val is greater than node
 - go left if val is less/equal than node
- ③ repeat 2 until leaf

```
1 def insertIntoBST(self, root, val) -> Optional[TreeNode]:  
2     if root is None:  
3         return TreeNode(val)  
4     node = root  
5     while True:  
6         if val > node.val:  
7             if node.right:  
8                 node = node.right  
9             else:  
10                 node.right = TreeNode(val)  
11                 break  
12         else:  
13             if node.left:  
14                 node = node.left  
15             else:  
16                 node.left = TreeNode(val)  
17                 break  
18     return root
```

$\mathcal{O}(\log n)$ time, $\mathcal{O}(1)$ space

Sorted Array to (balanced) BST

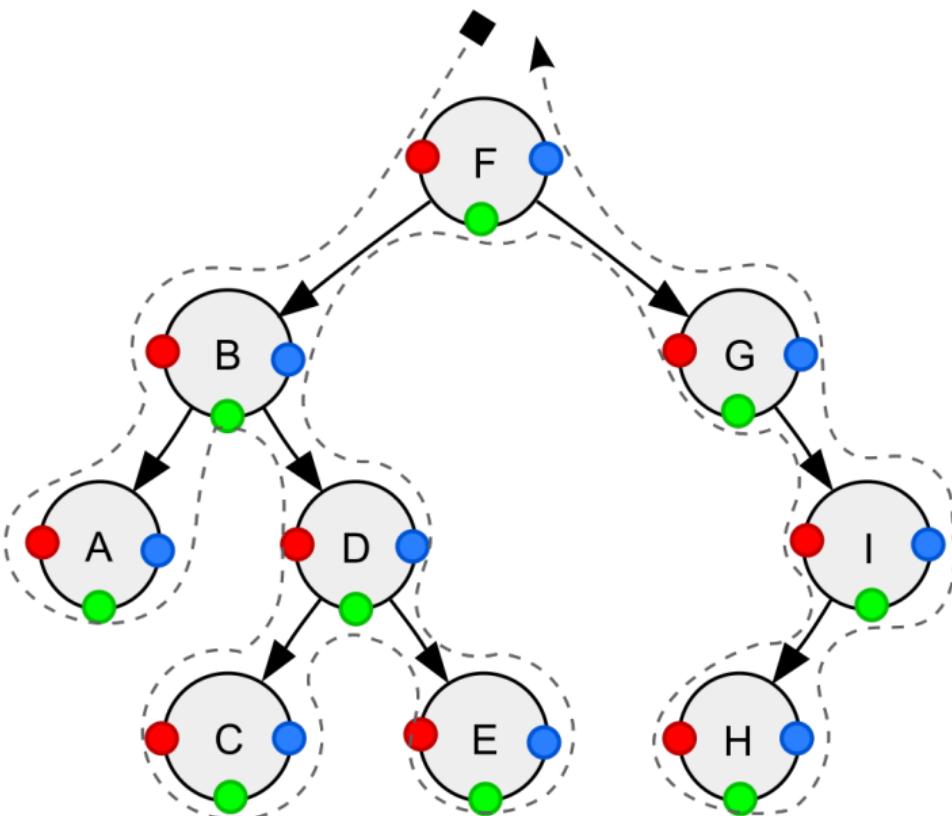
- ▶ take middle as root
- ▶ recurse into left half
- ▶ recurse into right half

```
1 def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:  
2     # Base condition...  
3     if len(nums) == 0:  
4         return None  
5     # set the middle node...  
6     mid = len(nums)//2  
7     # Initialise root node with value same as nums[mid]  
8     root = TreeNode(nums[mid])  
9     # Assign left subtrees  
10    root.left = self.sortedArrayToBST(nums[:mid])  
11    # Assign right subtrees.  
12    root.right = self.sortedArrayToBST(nums[mid+1:])  
13    # Return the root node...  
14    return root
```

$\mathcal{O}(\log n)$ time, $\mathcal{O}(\log n)$ space

Algorithmus: Tiefensuche auf Bäumen

- ▶ Tiefensuche / depth-first search (DFS)
- ▶ Pfad in die Tiefe und nicht in die Breite folgen
- ▶ Auf Bäumen drei Arten der Tiefensuche
 - Pre-order
 - In-order
 - Post-order



- ▶ Pre-order:
FBADCEGIH
- ▶ In-order:
ABCDEFGHI
- ▶ Post-order:
ACEDBHIGF

- ▶ Pre-order
 - ① Visit the current node.
 - ② Recursively traverse the current node's left subtree.
 - ③ Recursively traverse the current node's right subtree.
- ▶ In-order (sortiert bei BST)
 - ① Recursively traverse the current node's left subtree.
 - ② Visit the current node.
 - ③ Recursively traverse the current node's right subtree.
- ▶ Post-order
 - ① Recursively traverse the current node's left subtree.
 - ② Recursively traverse the current node's right subtree.
 - ③ Visit the current node.

► Pre-order

- ① Visit the current node.
- ② Recursively traverse the current node's left subtree.
- ③ Recursively traverse the current node's right subtree.

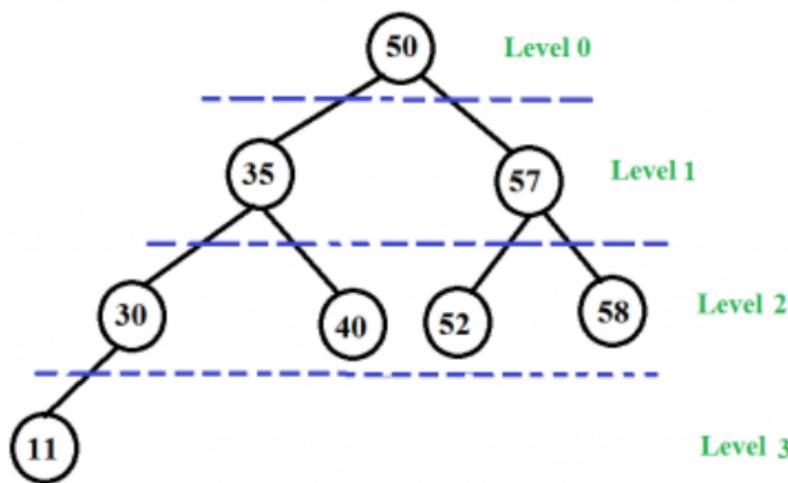
► In-order (sortiert bei BST)

- ① Recursively traverse the current node's left subtree.
- ② Visit the current node.
- ③ Recursively traverse the current node's right subtree.

► Post-order

- ① Recursively traverse the current node's left subtree.
- ② Recursively traverse the current node's right subtree.
- ③ Visit the current node.

Algorithmus: Breitensuche auf Bäumen /
Level Order Traversal



Level Order traversal:

50	35 57	30 40 52 58	11
L0	L1	L2	L3

- ① Erzeuge eine Liste von Listen
- ② Setze Tiefe = 0, starte bei der Wurzel
- ③ Hänge den aktuellen Knoten in die Liste der aktuellen Tiefe
- ④ Wiederhole 3,4,5 für den linken Kindsknoten mit Tiefe+1
- ⑤ Wiederhole 3,4,5 für den rechten Kindsknoten mit Tiefe+1

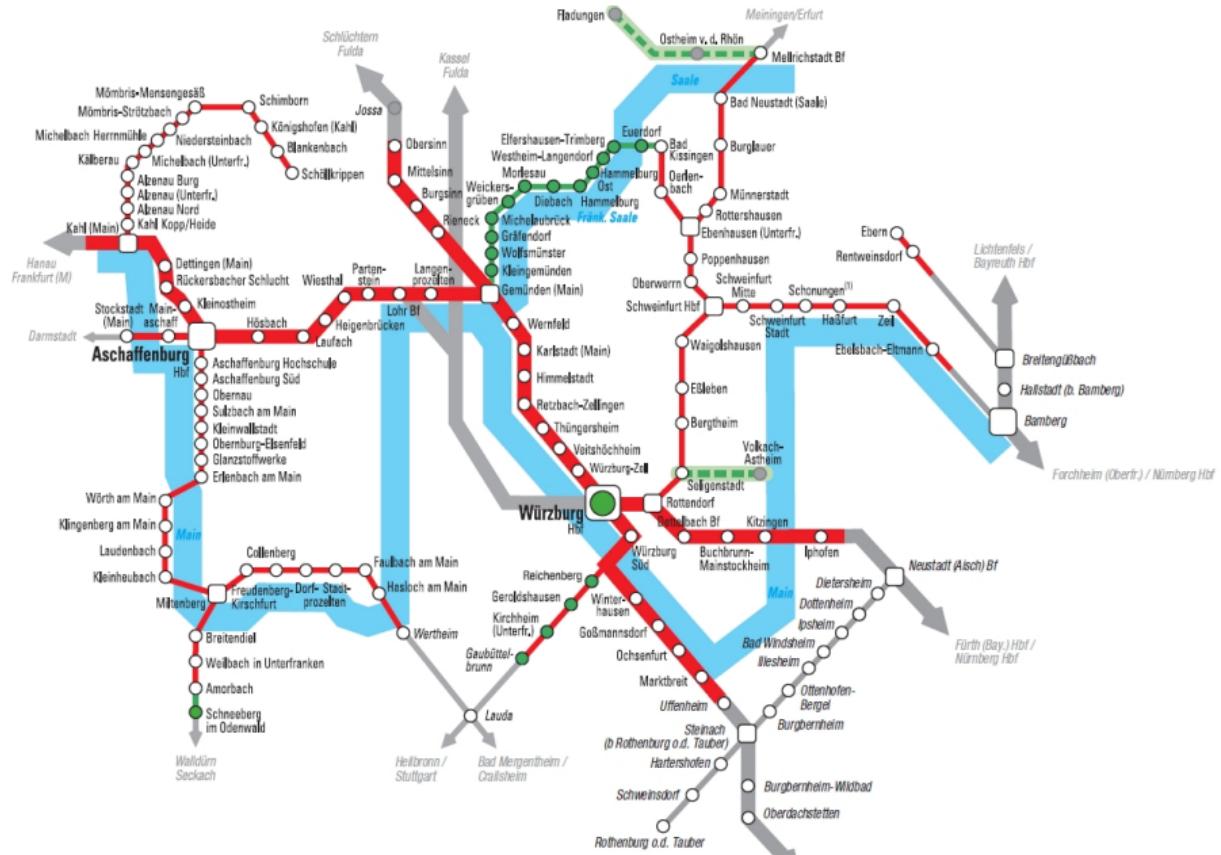
Übung

Implementieren Sie BFS Variante auf einem Baum

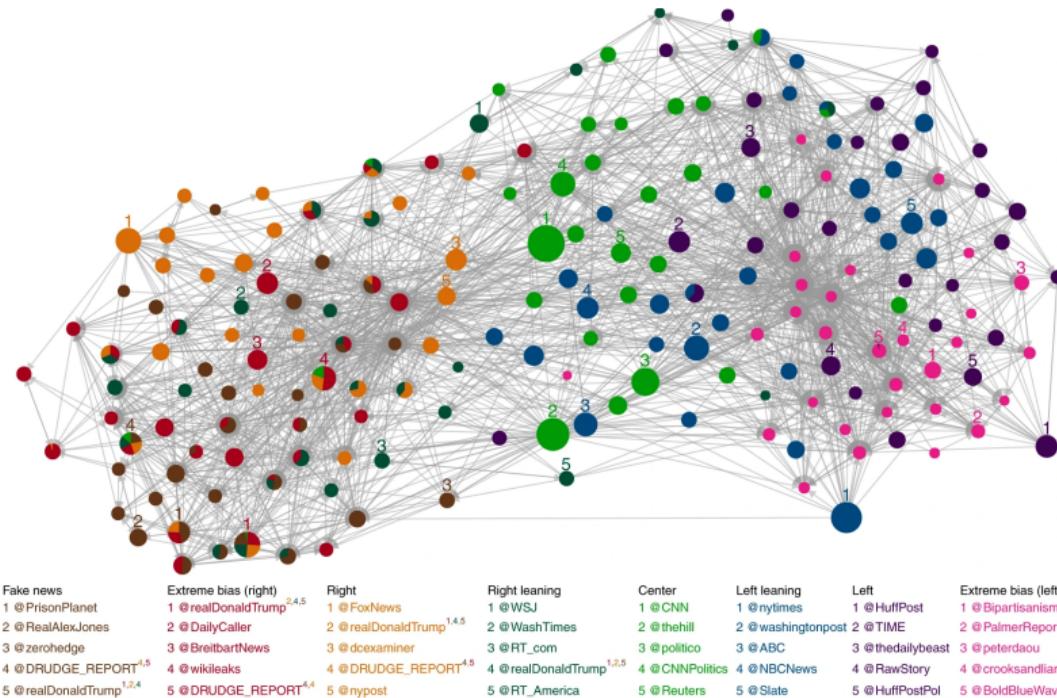
<https://leetcode.com/problems/binary-tree-level-order-traversal/>

Datenstruktur: Graphen

Schienennetzpläne

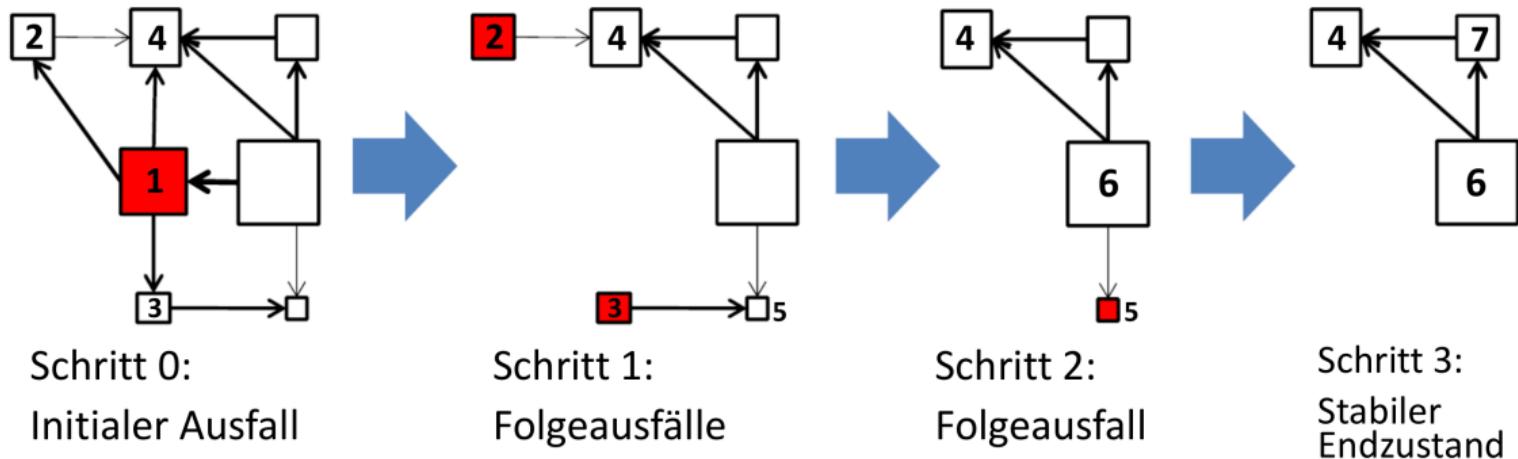


Retweet Netzwerk während 2016 US Wahl



Alexandre Bovet et al. "Influence of fake news in Twitter during the 2016 US presidential election". In: **Nature communications** 10.1 (2019), pp. 1–14.

Kreditausfälle



Tatiana von Landesberger et al. "Visual analysis of contagion in networks". In: **Information Visualization** (2013). URL: <http://dx.doi.org/10.1177/1473871613487087>.

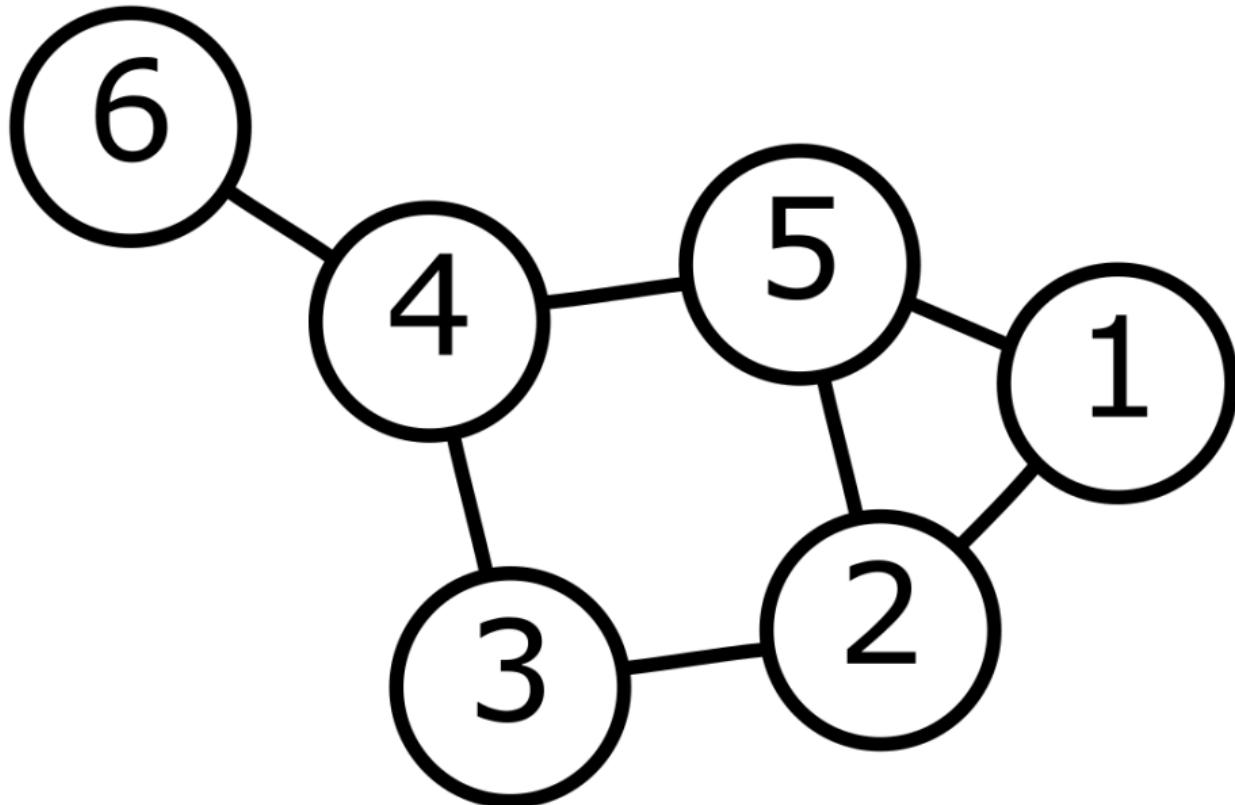
Das WWW als Netzwerk



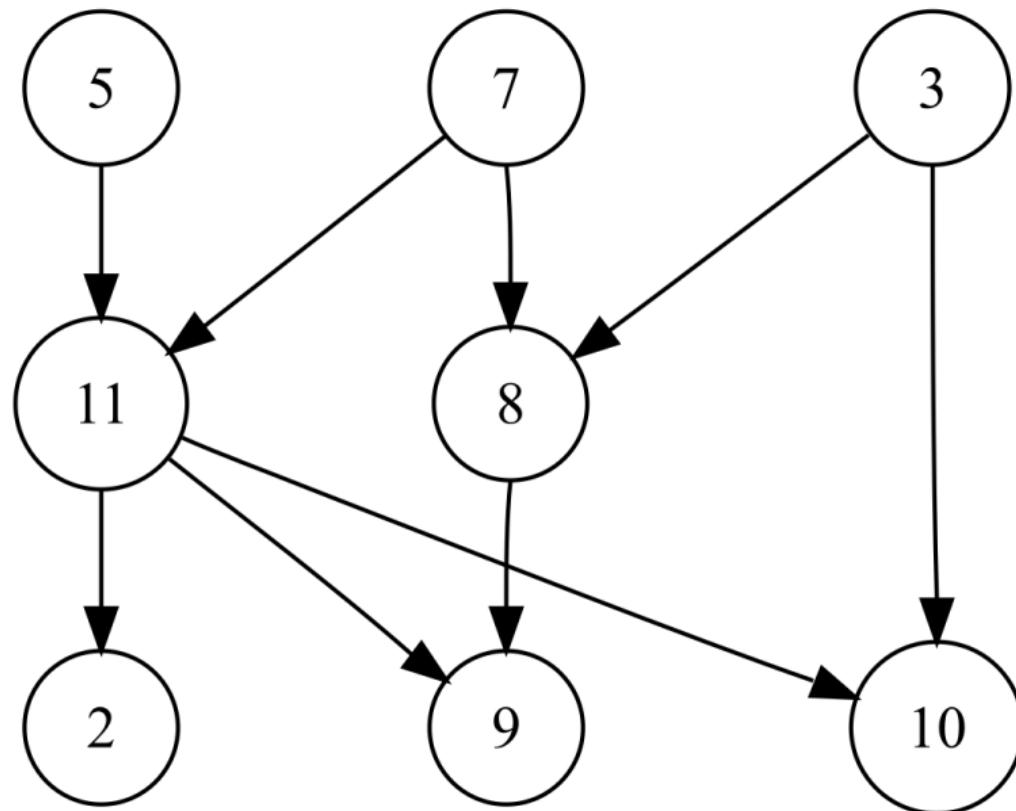
Lawrence Page et al. **The PageRank citation ranking: Bringing order to the web.**
Tech. rep. Stanford InfoLab, 1999.

- ▶ Graph G besteht aus Menge Knoten v und Kanten e
- ▶ Kanten können gerichtet oder ungerichtet sein
- ▶ Kanten können gewichtet oder ungewichtet sein
- ▶ Knoten und Kanten können beliebige weitere Datenattribute haben
- ▶ Verallgemeinerung von Bäume

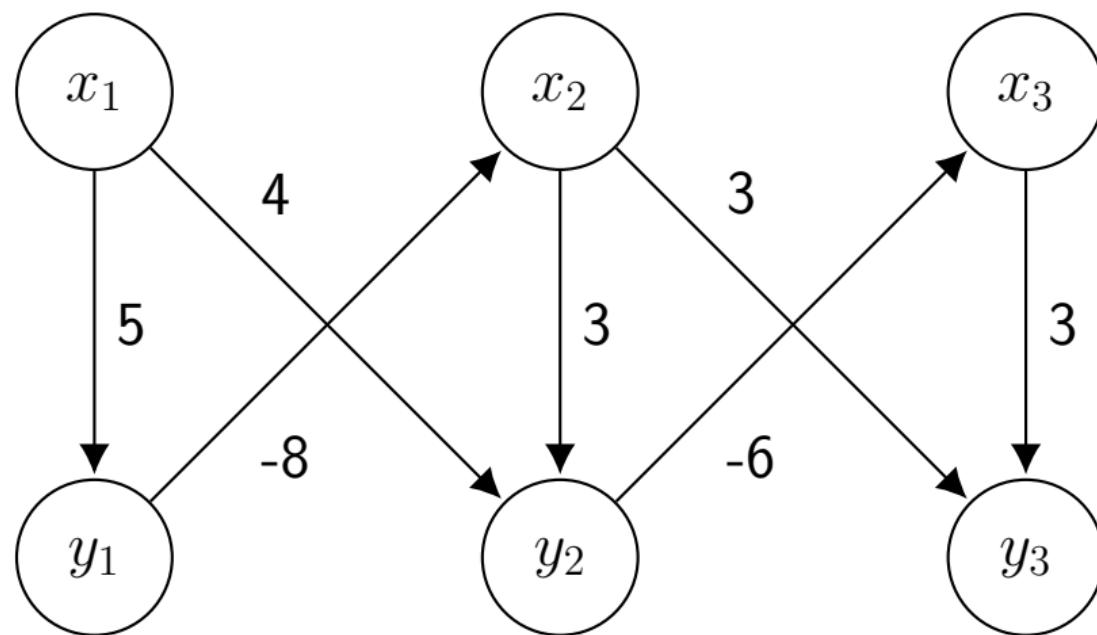
Ungerichteter Graph



Gerichteter Graph



Gerichteter, gewichteter Graph



Graph Fragestellungen

- ▶ Existiert ein Pfad zwischen A und B ?
- ▶ Was ist der kürzeste Pfad von A nach B ?
- ▶ Was ist der kürzeste Pfad von A zu allen anderen Knoten?
- ▶ Was ist die kürzeste Route um alle Knoten besucht zu haben?

- ▶ <https://www.youtube.com/watch?v=LFKZLXV0-Dg>
- ▶ <https://www.youtube.com/watch?v=x1VX7dXLS64>

Graph Datenstruktur

- ▶ Verschiedene Implementierung möglich
- ▶ Pointer-basiert (wie Linked List / Tree Node)
- ▶ Als Adjazenzliste
- ▶ Als Adjazenzmap
- ▶ Als Adjazenzmatrix

Live-Coding

Graph Datenstruktur in Python

- ▶ Vor- / Nachteile der Implementierungen?
- ▶ Sparse / Full
- ▶ Matrixoperationen / GPU
- ▶ Effizienze Sparse-Matrix Implementierungen

Übung

Adjazenzmatrix

<https://leetcode.com/problems/number-of-provinces>

- ▶ 5 Bonuspunkte für die Klausur
- ▶ 6 Bonusaufgaben
- ▶ Besten 5 Abgaben werden gewertet

Übungsaufgabe für Bonuspunkte

Eine der folgenden Leetcodes

- ▶ <https://leetcode.com/problems/clone-graph/>
- ▶ <https://leetcode.com/problems/find-if-path-exists-in-graph/>
- ▶ Reminder: Submission Screenshot + Quellcode txt (mit Name / Matrik)

Algorithmen und Datenstrukturen

Lecture 7

Prof. Dr. Maximilian Scherer
maximilian.scherer@dhw.de



SoSe 2025

Nachbesprechung

- ▶ Tiefensuche auf Bäumen
- ▶ Graphen

Breitensuche als Pseudocode

- ▶ Füge Startknoten v in eine Queue Q ein
- ▶ Solange Q nicht leer ist:
 - ① $u = Q.pop()$ # älteste element
 - ② markiere u als "besucht"
 - ③ füge alle unbesuchten Nachbarn von u in Q an

Laufzeit? $\mathcal{O}(|V| + |E|)$

Vorstellung Übungsaufgabe

- ▶ Kurzvorstellung Code
- ▶ Laufzeit / Speicher Analyse
- ▶ <https://leetcode.com/problems/clone-graph/>
- ▶ <https://leetcode.com/problems/find-if-path-exists-in-graph/>

Clone Graph

- ① Traverse all nodes (bfs / dfs)
- ② build hashmaps of copied nodes along the way
- ③ return hashmap entry for node 1

```

1 def cloneGraph(self, node: Node) -> Node:
2     if not node:
3         return node
4
5     q = deque([node])
6     clones ={node.val: Node(node.val, [])}
7     while q:
8         curr = q.popleft()
9         curr_clone = clones[curr.val]
10
11        for n in curr.neighbors:
12            if n.val not in clones:
13                clones[n.val] = Node(n.val, [])
14                q.append(neighbor)
15
16            curr_clone.neighbors.append(clones[n.val])
17    return clones[1]

```

$\mathcal{O}(|V| + |E|)$ time, $\mathcal{O}(|V| + |E|)$ space

```
1 def cloneGraph(self, node: Node) -> Node:
2     if not node:
3         return node
4
5     clones = {}
6
7     def clone(node):
8         if node in clones:
9             return clones[node]
10
11        copy = Node(node.val)
12        clones[node] = copy
13
14        for n in node.neighbors:
15            copy.neighbors.append(clone(n))
16        return copy
17
18    return clone(node)
```

$\mathcal{O}(|V| + |E|)$ time, $\mathcal{O}(|V| + |E|)$ space

Does Path Exist?

- ▶ Create adjacency list / map
- ▶ do bfs / dfs
- ▶ return true if dest found, else return false at end

```
1 def validPath(self, n: int, edges: List[List[int]], s, dest):
2     adjlist = {}
3     for i, j in edges:
4         if i not in adjlist:
5             adjlist[i] = []
6         if j not in adjlist:
7             adjlist[j] = []
8         adjlist[i].append(j)
9         adjlist[j].append(i)
10
11    visited = set()
12    q = deque()
13    q.append(s)
14    visited.add(s)
15
16    while q:
17        node = q.popleft()
18        if node == dest:
19            return True
20        for neighbor in adjlist[node]:
21            if neighbor not in visited:
22                q.append(neighbor)
23                visited.add(neighbor)
24
25    return False
```

Datastructure: HEAP

- ▶ HEAP ist eine Binärbaum-basierte Datenstruktur
- ▶ Kann effizient als Array gespeichert werden, weil folgende Eigenschaft gelten muss
- ▶ Für jeden Knoten C gilt, dass der Wert von jedem Parent-Knoten P größer (bzw. kleiner bei MINHEAP) oder gleich sein muss, als der Wert von C
- ▶ Entspricht Priority Queue

HEAPs in Python

```
1 import heapq
2
3 l = [5, 7, 9, 1, 3]
4
5 # using heapify to convert list into heap
6 heapq.heapify(l) #inplace
7
8 print(f"The created heap is : {l}")
9
10 # using heappush() to push elements into heap
11 heapq.heappush(l, 4)
12 print(f"modified heap is : {l}")
13
14 # using heappop() to pop smallest element
15 print(heapq.heappop(l))
```

- ▶ heapify sortiert Liste so um, dass HEAP gilt
- ▶ heappush Element hinzufügen, HEAP sicherstellen
- ▶ heappop Root entfernen, HEAP sicherstellen
- ▶ Laufzeit von push / pop?
- ▶ <https://visualgo.net/en/heap>

Übung

[https://leetcode.com/problems/
kth-largest-element-in-an-array/](https://leetcode.com/problems/kth-largest-element-in-an-array/)

Kürzeste Pfade

- ▶ Was ist der kürzeste Pfad von A nach B ?
 - Breitensuche (bfs) für ungewichtete Graphen
 - Dijkstra's Algorithm für gewichtete Graphen (berechnet kürzesten Weg von A zu allen Knoten)
 - A* Algorithm für gewichtete Graphen (schnell, heuristisch, oft in Games genutzt, behandeln wir hier nicht)
- ▶ <https://www.youtube.com/watch?v=pVfj6mxhdMw>

Übung

<https://leetcode.com/problems/network-delay-time/>

Klausur

- ▶ Eine “bekannte” Leetcode Aufgabe (aus Übung / Vorlesun)
 - “easy” oder “medium”, leicht abgeändert oder ähnlich (z.B. Two Sum 1 und Two Sum 2)
 - 30 Punkte, 15-20 für Idee/Code/Erklärung, 10 für Analyse (nachvollziehbar), ggf. 5 Punkte für zusätzliche Einschränkung
- ▶ Eine neue Leetcode Aufgabe
 - “easy” oder “medium”
 - ggf. leicht abgeändert
 - 30 Punkte, 15-20 für Idee/Code/Erklärung, 10 für Analyse (nachvollziehbar), ggf. 5 Punkte für zusätzliche Einschränkung

Klausurvorbereitung

- ▶ Grundlegende Algorithmen und Datenstrukturen verstehen / anwenden können / üben (das auf Spickzettel schreiben)
- ▶ Alle Übungsaufgaben (Vorlesung und Übung) erneut durchgehen
 - Selbst lösen
 - Andere Lösungen und Erklärungen dazu anschauen
- ▶ Weitere Leetcode Aufgaben zu unseren Vorlesungsthemen üben

- ▶ Implementieren Sie einen entsprechenden Algorithmus für den obigen Funktionskopf in Pseudo-Code / Python / andere Programmiersprache und kommentieren / erklären Sie Ihre Herangehensweise (15 Punkte)
- ▶ Implementieren Sie Ihren Algorithmus so, dass dieser ein sinnvolles Early Stopping Kriterium einsetzt (also hier nicht alle Moves evaluieren muss). (5 Punkte)
- ▶ Analysieren Sie Laufzeit und Speicherbedarf in \mathcal{O} -Notation. Die Berechnung / Analyse muss nachvollziehbar sein. Sie können dies z.B. direkt als Kommentare im Code tun. (10 Punkte)

Fragen

- ▶ Fragen zur Klausur
- ▶ Fragen zu unseren Themen
- ▶ Konkrete Rückfragen zu Leetcode Aufgaben