Programmierkurs

Vorlesung 10

M.Sc. Laslo Hunhold

Department Mathematik/Informatik
Abteilung Informatik
Universität zu Köln

25. Januar 2021

Ankündigungen

- Letzte Vorlesung (11) und letztes Übungsblatt (11) am 01.02.
- Letzte Übungswoche vom 08.02. bis 12.02. (Besprechung Übungsblatt 11)
- Sonderübungen am Montag, den 01.02., und Mittwoch, den 03.02., jeweils um 14 Uhr (thematisch identisch)

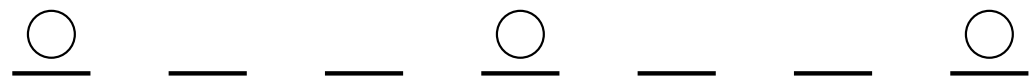
Letzte Vorlesung

- Annotationen
- Export/Import von Packages (`import`, Programmhierarchie)
- Klassenbibliothek/API (`java.lang`, `java.io`, `java.util`, etc.)
- Throwables (Exceptions/Errors, `try...catch`, `throw`, `throws`, Runtime Exceptions vs. Checked Exceptions)
- Dateien lesen und schreiben (`File`, `FileReader`, `FileWriter`)

Scheduling (Sequenzplanung)

Beispiel

- 7 Sitzplätze sollen möglichst optimal belegt werden
- Regel: Zwischen jeder Person müssen 2 Plätze frei bleiben
- Beispielbelegungen

0. 
1. 
2. 

- Belegungsziele
  - „Kardinalitätsmaximal“: Die Zuweisung ist maximal groß (Beispiel 2)
  - „Inklusionsmaximal“: Die Zuweisung ist nicht erweiterbar (Beispiel 1)
  - Beispiel 0 ist weder inklusions- noch kardinalitätsmaximal
  - Kardinalitätsmaximal $\Rightarrow$ inklusionsmaximal
  - Inklusionsmaximal $\not\Rightarrow$ kardinalitätsmaximal

Interval Scheduling

Problem

Gegeben

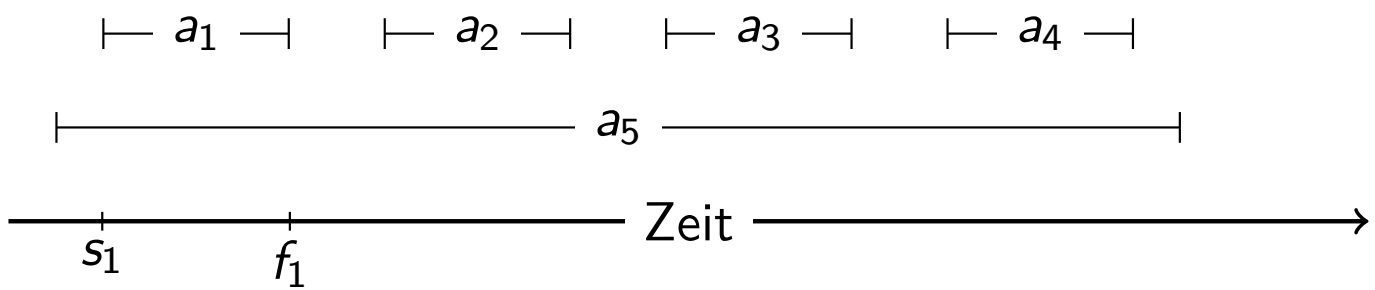
- Eine Ressource (z.B. Raum, Prozessor, Mietwagen)
- Eine Menge A von Anfragen für diese Ressource, jeweils mit Start- und Endzeitpunkt
- Die Ressource kann nur von einem gleichzeitig verwendet werden

Gesucht

- Eine (kardinalitäts)maximale Teilmenge S von A, die paarweise zeitlich überschneidungsfrei („kompatibel“) ist

Formalisierung

- Sei $A := \{a_1, \ldots, a_n\} := \{(s_1, f_1), \ldots, (s_n, f_n)\}$ die Menge der Anfragen a_i mit Startzeitpunkt s_i und Endzeitpunkt f_i
- $S \subseteq A$ kompatibel $:\Leftrightarrow \forall_{(s,f)\neq(\tilde{s},\tilde{f})\in S} : (\tilde{s} \geq s \rightarrow \tilde{s} \geq f)$
- Ziel: Finde $S \subseteq A$ kompatibel, sodaß $|S|$ maximal



Lösung (Greedy-Interval-Scheduling-Algorithmus)

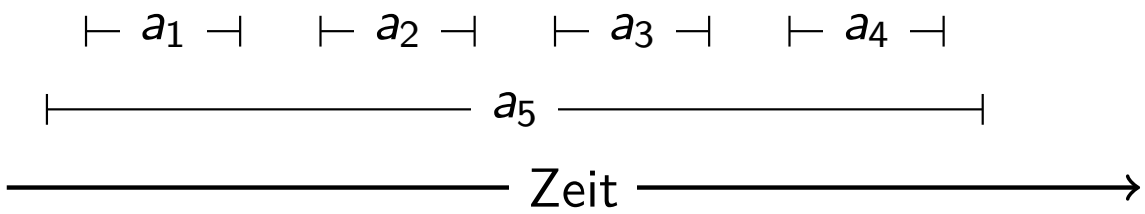
input : Menge der Anfragen
$A := \{a_1, \ldots, a_n\} := \{(s_1, f_1), \ldots, (s_n, f_n)\}$
output: Menge der akzeptierten Anfragen $S \subseteq A$

$S \leftarrow \emptyset$;
while $A \neq \emptyset$ **do**
  $a \leftarrow$ waehle_aus(A);
  $A \leftarrow A \setminus \{a\}$; /* entferne a aus A */
  $S \leftarrow S \cup \{a\}$; /* füge a zu S hinzu */
  $A \leftarrow A \setminus \{\tilde{a} \in A \mid \tilde{a}$ überlappt $a\}$;
end
return S;

Wie definieren wir waehle_aus() (i.e. wie wählen wir unsere nächste Anfrage aus A)?

Anfragenauswahl (Frühester Startzeitpunkt)

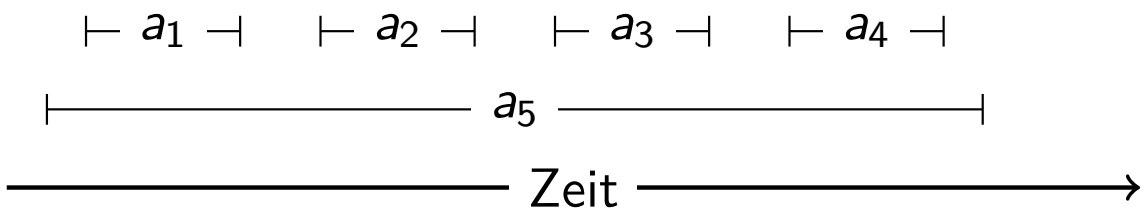
- Wähle $a = (s, f) \in A$ mit s minimal
- Motivation: Ressource soll möglichst vollbelegt sein
- Beispiel 0



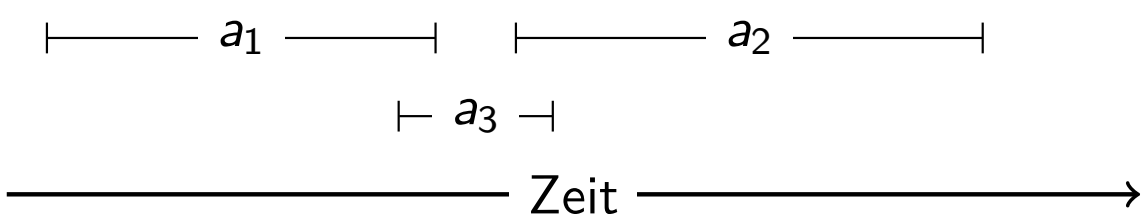
  - a_5 hat den frühesten Startzeitpunkt
  - Lösung $S = \{a_5\}$ kleiner als maximale Lösung $\{a_1, a_2, a_3, a_4\}$
- Auswahlmethode nicht zielführend

Anfragenauswahl (Kürzestes Intervall)

- Wähle $a = (s, f) \in A$ mit $f - s$ minimal
- Motivation: Ein kurzes Intervall ist leichter unterzubringen
- Beispiel 0



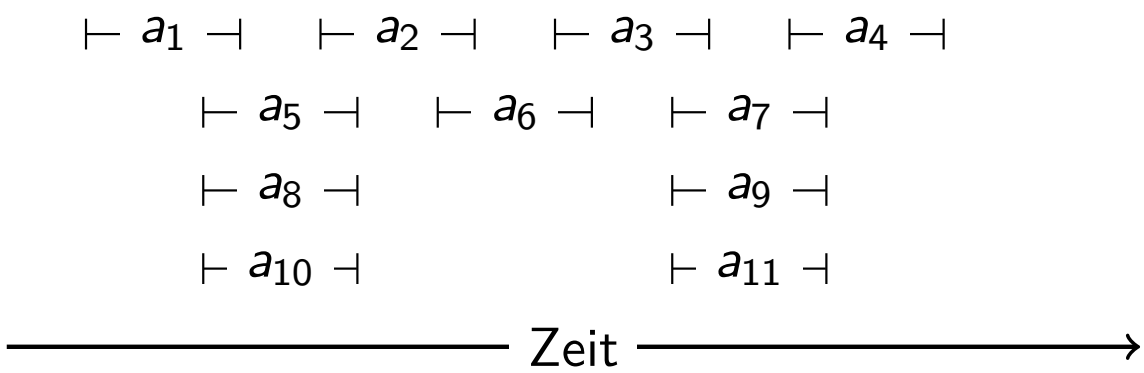
  - a_1 ist eines der kürzesten Intervalle
  - Lösung $S = \{a_1, a_2, a_3, a_4\}$ entspricht der maximalen Lösung
- Beispiel 1



  - a_3 ist das kürzeste Intervall
  - Lösung $S = \{a_3\}$ kleiner als die maximale Lösung $\{a_1, a_2\}$
- Auswahlmethode nicht zielführend

Anfragenauswahl (Minimale Überlappung)

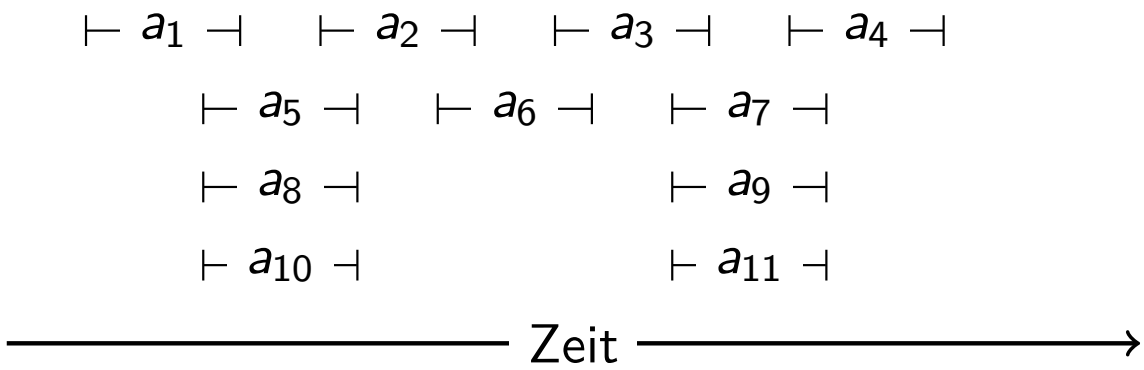
- Wähle $a \in A$ mit $|\{\tilde{a} \in A \mid \tilde{a} \text{ überlappt } a\}|$ minimal
- Motivation: So wenige „Abzüge“ aus A pro Iteration wie möglich
- Beispiele 0 und 1 funktionieren (Übungsaufgabe 11-0)
- Beispiel 2



  - a_6 hat die wenigsten Überlappungen, eliminiert a_2 und a_3
  - Danach hat a_7 u.a. die wenigsten Überlappungen, eliminiert a_4, a_9 und a_{11}
  - Danach hat a_1 u.a. die wenigsten Überlappungen, eliminiert a_5, a_8 und a_{10}
  - Lösung $S = \{a_1, a_6, a_7\}$ kleiner als die maximale Lösung $\{a_1, a_2, a_3, a_4\}$
- Auswahlmethode nicht zielführend

Anfragenauswahl (Frühester Endzeitpunkt)

- Wähle $a = (s, f) \in A$ mit f minimal
- Motivation: Ressource schnellstmöglich wieder freigeben
- Beispiele 0 und 1 funktionieren (Übungsaufgabe)
- Beispiel 2



  - a_1 hat den frühesten Endzeitpunkt, eliminiert a_5, a_8 und a_{10}
  - Danach hat a_2 den frühesten Endzeitpunkt, eliminiert a_6
  - Danach hat a_3 den frühesten Endzeitpunkt, eliminiert a_7, a_9 und a_{11}
  - Lösung $S = \{a_1, a_2, a_3, a_4\}$ entspricht der maximalen Lösung
- Auswahlmethode zielführend für drei Beispiele (kein Beweis, aber Indiz)
- Definieren waehle_aus() über den frühesten Endzeitpunkt

Terminierung

- Proposition: Der Greedy-Interval-Scheduling-Algorithmus terminiert
- Beweis:
  - In jeder Iteration wird mindestens eine Anfrage (nämlich das ausgewählte a) aus A entfernt
  - Nach spätestens $|A| = n$ Durchläufen ist A also leer
  - Wenn A leer ist, terminiert der Algorithmus □

Kompatibilität

- Proposition: S ist kompatibel
- Beweis:
  0. Seien $p, q \in S$
  1. Angenommen q überlappt sich mit p
  2. O.B.d.A. wurde p vor q zu S hinzugefügt $\overset{1}{\Rightarrow}$ q wurde danach aus A gelöscht $\Rightarrow q \notin S \lightning \overset{0}{\Rightarrow} \neg 1$
  3. $2 \Rightarrow q$ überlappt sich nicht mit $p \Rightarrow S$ ist kompatibel $\square$

Optimalität

- Voraussetzungen:
  - Seien o.B.d.A. die a_is so nummeriert, daß mit $k \in \{1, \ldots, n\}$ $S = \{a_1, \ldots, a_k\} = \{(s_1, f_1), \ldots, (s_k, f_k)\}$ gilt
  - Sei $O := \{\tilde{a}_1, \ldots, \tilde{a}_m\} := \{(\tilde{s}_1, \tilde{f}_1), \ldots, (\tilde{s}_m, \tilde{f}_m)\}$ mit $m \geq k$ eine optimale Lösungsmenge mit $\forall_{i,j \in \{1,\ldots,m\}} : (i < j \rightarrow \tilde{f}_i < \tilde{f}_j)$ (nach Endzeit sortiert)
- Lemma: $\forall_{i \in \{1,\ldots,k\}} : f_i \leq \tilde{f}_i$
- Beweis: Beweis durch vollständige Induktion über $i \in \{1, \ldots, k\}$
  - Induktionsanfang ($i = 1$): Algorithmus wählt a_1 nach frühestem Endzeitpunkt, nicht verbesserbar $\Rightarrow f_1 \leq \tilde{f}_1$
  - Induktionsvoraussetzung: $f_i \leq \tilde{f}_i$ gelte für $i \in \{1, \ldots, k-1\}$ fest aber beliebig
  - Induktionsschritt ($i \in \{1, \ldots, k-1\}, i \mapsto i+1$):
    0. O optimal $\Rightarrow$ O kompatibel $\Rightarrow \tilde{f}_i \leq \tilde{s}_{i+1} \overset{IV}{\Rightarrow} f_i \leq \tilde{s}_{i+1} \Rightarrow a_i$ und $\tilde{a}_{i+1}$ überlappen sich nicht $\Rightarrow \tilde{a}_{i+1}$ ist nach a_i-Iteration noch vorhanden
    1. Algorithmus wählt a_{i+1} nach frühestem Endzeitpunkt $\overset{0}{\Rightarrow} f_{i+1} \leq \tilde{f}_{i+1}$ □
- Proposition: S ist (kardinalitäts)maximal, also optimal
- Beweis:
  0. Angenommen S nicht (kardinalitäts)maximal $\Leftrightarrow k < m$
  1. O optimal $\Rightarrow$ O kompatibel $\overset{0}{\Rightarrow} \exists_{\tilde{a}_{k+1}} : \tilde{f}_k \leq \tilde{s}_{k+1}$
  2. Lemma $\Rightarrow f_k \leq \tilde{f}_k \overset{1}{\Rightarrow} f_k \leq \tilde{s}_{k+1} \Rightarrow a_k$ und $\tilde{a}_{k+1}$ überlappen sich nicht $\Rightarrow$ $\tilde{a}_{k+1}$ ist nach a_k-Iteration noch vorhanden $\Rightarrow S$ unvollständig $\lightning \Rightarrow \neg 0$ □

Bemerkungen

- Nutzen für Alltag (Anfragen nach frühestem Endzeitpunkt abzuarbeiten ist bewiesenermaßen optimal)
- Erweiterungen
  - Online Interval Scheduling
    - Anfragen treffen dynamisch ein
    - Anfragen verschwinden nach einer gewissen Zeit
    - Beispiel: Restaurant
  - Weighted Interval Scheduling
    - Anfragen sind gewichtet (i.e. priorisiert)
    - Beispiel: Supercomputer (Anfrage von Professor, Doktorand, Student, etc.)

Minimize Maximum Lateness

Problem

Gegeben

- Eine Menge A von Aufgaben, jeweils mit (Arbeits)dauer und Fälligkeitsdatum
- Man kann nur eine Aufgabe gleichzeitig erledigen
- Eventuell sind nicht alle Aufgaben bis zum Fälligkeitsdatum erledigbar, Überziehung („Lateness“) sollte möglichst vermieden werden

Gesucht

- Ein Startzeitpunkt für jede Aufgabe, sodaß die Aufgabenerledigung überschneidungsfrei und möglichst pünktlich ist

Formalisierung

- Sei $A := \{a_1, \ldots, a_n\} := \{(d_1, f_1), \ldots, (d_n, f_n)\}$ die Menge der Aufgaben a_i mit Dauer d_i und Fälligkeitsdatum f_i
- Sei $S := \{(a_1, s_1), \ldots, (a_n, s_n)\}$ die Lösungsmenge der Termine (a_i, s_i) mit der Aufgabe a_i und Startzeitpunkt s_i
- „Lateness“ eines Termins (a_i, s_i)

$$\ell(a_i, s_i) := \begin{cases} s_i + d_i - f_i & s_i + d_i > f_i \\ 0 & s_i + d_i \leq f_i \end{cases}$$

- „Maximum Lateness“ einer Terminmenge $\tilde{S}$

$$L(\tilde{S}) := \max_{(a,s) \in \tilde{S}} \ell(a, s)$$

- Finde S, sodaß $L(S)$ minimal (i.e. „Minimize Maximum Lateness“)

Lösung (Greedy-Minimize-Maximum-Lateness-Algorithmus)

input : Menge der Aufgaben
$A := \{a_1, \ldots, a_n\} := \{(d_1, f_1), \ldots, (d_n, f_n)\}$
Beginn der Arbeitszeit b
output: Menge der Termine S

$S \leftarrow \emptyset$;
$s \leftarrow b$; /* setze Startzeit auf Beginn */
while $A \neq \emptyset$ **do**
  $a_i \leftarrow \arg\min_{a_k \in A} f_k$; /* a_i frühestes Fälligkeitsdatum */
  $A \leftarrow A \setminus \{a_i\}$; /* entferne a_i aus A */
  $S \leftarrow S \cup \{(a_i, s)\}$; /* füge (a_i, s) zu S hinzu */
  $s \leftarrow s + d_i$; /* erhöhe Startzeit um Dauer von a_i */
end
return S;

- Algorithmus minimiert die Maximum Lateness (i.e. S ist eine optimale Lösung)
- Unterstreicht Nützlichkeit von Greedy-Ansätzen (vgl. auch Interval Scheduling)

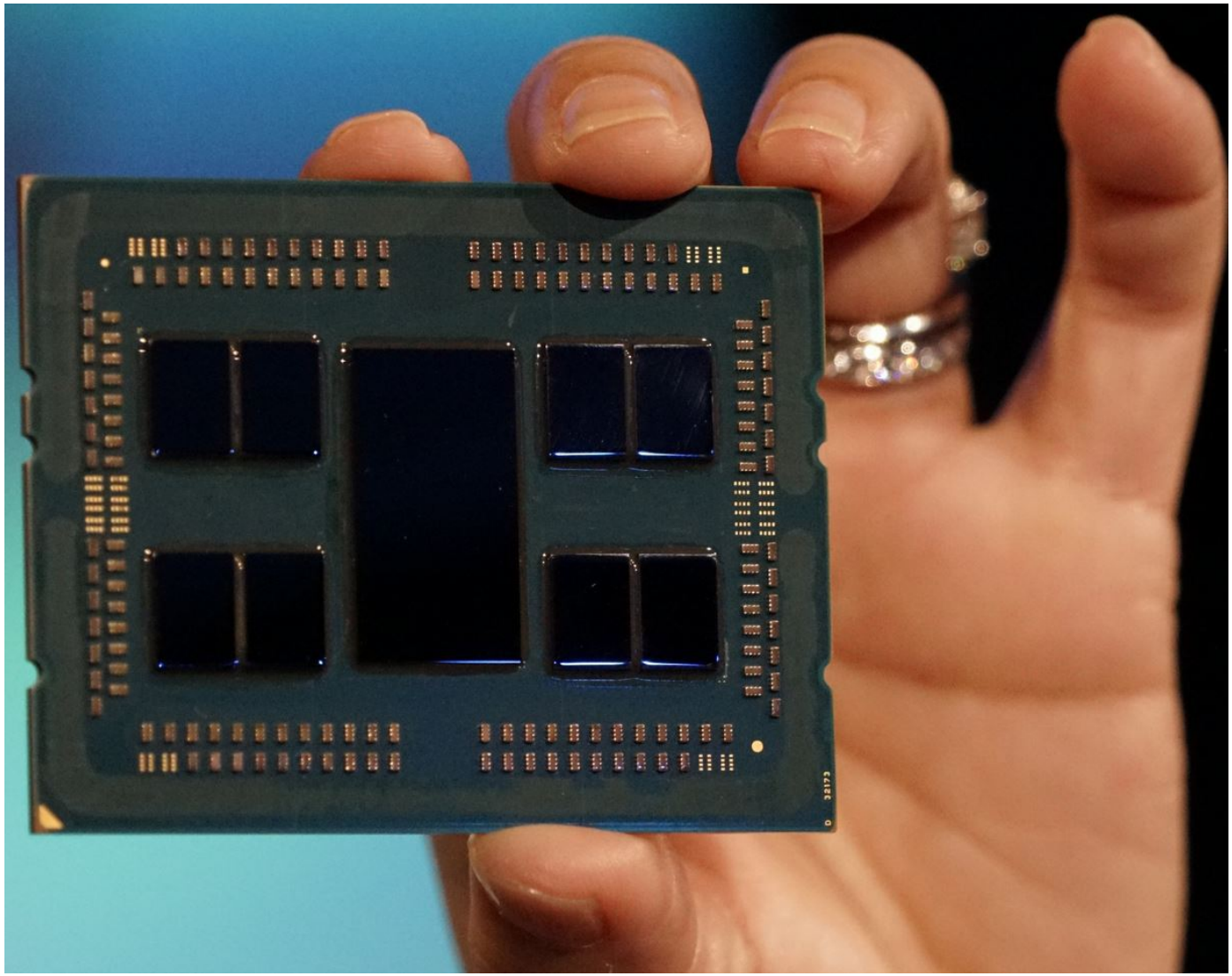


Abbildung: AMD EPYC Rome 7H12 (64 Kerne, 128 Threads) (2019) (Quelle: Tom's Hardware)