



Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2023 · tcs.uni-frankfurt.de/algo1/ · 2023-07-06 · ed6968a








Einführung und Peaks (Woche 1)

Eigenständige Vorbereitung:

Lies  CLRS Kapitel 1 und schau dir das  Video der Woche an. Beantworte dabei die folgenden Leitfragen:

- Was ist der Unterschied zwischen einem Algorithmus und einem Programm?
- Was sagt die Laufzeit eines Algorithmus über das zugehörige Programm aus?
- Was sind Hügel und wie findet man auf geschickte Weise einen?

Zeichenlegende:

-  Schriftliche Aufgabe, die du fristgerecht in Moodle abgibst. In der Klausur wirst du alle Aufgaben schriftlich bearbeiten, daher ist das Feedback der Tutoren wichtig, damit du deine Schreibfähigkeiten verbessern kannst.
-  Diese Art von Aufgabe musst du sicher können, um die Klausur zu bestehen.
-  Diese Art von Aufgabe musst du weitgehend können, um die Klausur zu bestehen.
-  Diese Art von Aufgabe musst du können, um eine gute Note zu erhalten.
-  Diese Aufgabe ist als Knobelspaß gedacht, der das algorithmische Verständnis vertieft.

Aufgabe 1.1 (Schleifen 👉). Schau Dir das Codefragment an (entweder in Python oder in Java/C/C++). Was geben die Funktionen loop1, loop2, loop3, loop4 jeweils zurück, wenn die Eingabe $n = 4$ ist? Was bei Eingabe $n = 10$? Was bei Eingabe $n = 1000$? Was geben sie als Funktion von n zurück?

```

1 # Python
2
3 def loop1(n):
4     x = 0
5     for i in range(n):
6         for j in range(n):
7             x += 1
8     return x
9
10 def loop2(n):
11     x = 0
12     for i in range(n):
13         x += 1
14     for j in range(n):
15         x += 1
16     return x
17
18 def loop3(n):
19     x = 0
20     for i in range(n):
21         if (i == n-1):
22             for j in range(n):
23                 x += 1
24     return x
25
26 def loop4(n):
27     x = 0
28     for i in range(n):
29         for j in range(i,n):
30             x += 1
31     return x
32
33
34
1 /* Java/C/C++ */
2
3 int loop1(int n) {
4     int x = 0;
5     for(int i = 0; i < n; i++) {
6         for(int j = 0; j < n; j++) x++;
7     }
8     return x;
9 }
10
11 int loop2(int n) {
12     int x = 0;
13     for(int i = 0; i < n; i++) x++;
14     for(int j = 0; j < n; j++) x++;
15     return x;
16 }
17
18 int loop3(int n) {
19     int x = 0;
20     for(int i = 0; i < n; i++) {
21         if (i == n-1)
22             for(int j = 0; j < n; j++)
23                 x++;
24     }
25     return x;
26 }
27
28 int loop4(int n) {
29     int x = 0;
30     for(int i = 0; i < n; i++) {
31         for(int j = i; j < n; j++) x++;
32     }
33     return x;
34 }

```

Aufgabe 1.2 (Rekursion und Iteration 👍). Eine Funktion ist *rekursiv*, wenn sie sich selbst aufruft. Zum Beispiel:

```

1 # Python
2
3 def f(A, n):
4     if (n == 0):
5         return 0
6     else:
7         return f(A, n-1) + A[n-1]

```

```

1 /* Java/C/C++ */
2
3 int f(int[] A, int n) {
4     if(n == 0) return 0;
5     else return f(A, n-1) + A[n-1];
6 }

```

- Was liefert $f(A, n)$ zurück, wenn A ein Array ist, das n Zahlen enthält?
- Kann man $f(A, n)$ auch *iterativ* machen? Das heißt, kann man den Code so umschreiben, dass die Funktion sich nicht selbst aufruft, aber dasselbe Ergebnis liefert? Wie?

Aufgabe 1.3 (Coronatests). Das Gesundheitsamt schickt euch eine Gruppe von 128 Personen mit der gesicherten Information, dass genau eine davon mit SARS-CoV-2 infiziert ist. Aber wer ist es? Euer Testzentrum hat nur eine PCR-Maschine zur Verfügung und ein Durchlauf auf einer Probe dauert eine Stunde. Ihr wollt das Ergebnis aber so schnell wie möglich wissen. Da das PCR-Verfahren hochsensitiv ist, könnt ihr problemlos Abstriche von mehreren Personen zu einer Probe zusammenlegen, um zu testen, ob mindestens eine der gewählten Personen positiv ist.

- 👍 Wie findet man den positiven Fall in 7 Stunden?
- 👍 Wie viele Stunden braucht ihr, um n Personen zu testen anstatt 128?
- 🔑 Unerwarteterweise finden sich in einem Nebenraum deines Testzentrums plötzlich $k - 1$ verstaubte, aber funktionsbereite PCR-Maschinen, ihr könnt also k Proben pro Stunde testen. Wie viele Stunden braucht ihr in dieser Situation, um den positiven Fall zu finden?

Aufgabe 1.4 (Zombieduelle 🌈). Ihr habt eine Armee von n hirnlosen Zombies und wollt den stärksten und den schwächsten finden. Ihr könnt zwei Zombies in einen Käfig sperren und ein Stück Fleisch zwischen sie werfen, um umgehend herauszufinden, wer von den beiden der Stärkere ist. Da ihr die massenhafte Fleischproduktion aus ethischen Gründen ablehnt, wollt ihr die Zahl der Duelle so klein wie möglich halten.

- Wie findet man den stärksten Zombie mit höchstens $n - 1$ Duellen?
- (schwer) Wie findet man sowohl den stärksten als auch den schwächsten Zombie mit höchstens $3n/2$ Duellen?
- (sehr schwer) Wie findet man sowohl den stärksten als auch den zweitstärksten Zombie mit höchstens $n + \log_2 n$ Duellen?

Aufgabe 1.5 (Ameisen auf dem Stock 🌈, sehr schwer). Auf einem 100 cm langen Stock befinden sich 100 Ameisen. Zu Beginn wird jede Ameise auf dem Stock positioniert und entweder nach links oder nach rechts ausgerichtet. Dann fangen alle Ameisen gleichzeitig an, sich zu bewegen. Jede Ameise bewegt sich exakt 1 cm pro Sekunde. Wenn eine Ameise auf eine ihr entgegen kommende Ameise trifft, drehen sich beide Ameisen sofort um und laufen in die andere Richtung weiter. Wenn eine Ameise eines der beiden Stockenden erreicht, fällt sie runter. Wie lange dauert es maximal (über alle möglichen Startkonfigurationen), bis alle Ameisen vom Stock gefallen sind?

Aufgabe 1.6 (Hügel). Sei $A = [2, 1, 3, 7, 3, 11, 1, 5, 7, 10]$.

- 👉 (einfach) Finde alle Hügel von A .
- 👉 (einfach) Welche Hügel werden von den beiden Linearzeitalgorithmen jeweils ausgegeben?
- 👉 Gib die Sequenz von rekursiven Aufrufen an, die der rekursive Algorithmus produziert. Nimm hierfür zunächst an, dass der Algorithmus immer die linke Hälfte des Arrays wählt, wenn beide Richtungen möglich wären. Gib anschließend alle möglichen Sequenzen von rekursiven Aufrufen an, die der Algorithmus machen kann, wenn er jedes Mal eine zufällige gültige Richtung wählt.

Aufgabe 1.7 (Täler 👉). Überlegt euch ein *Talproblem* als Gegenstück zum Hügelproblem. Stellt eine präzise Definition des Talproblems auf.

Aufgabe 1.8 (Algorithmen und Anwendungen 👉). Diskutiert:

- Kann ein algorithmisches Problem von verschiedenen Algorithmen gelöst werden?
- Kann ein und derselbe Algorithmus mehr als ein algorithmisches Problem lösen?
- Kann ein abstrakter Datentyp von verschiedenen Datenstrukturen implementiert werden?
- Gib ein Problem in der echten Welt an, für das man zwingend die optimale Lösung braucht. Gib ebenso ein Problem an, wo eine approximative Lösung ausreicht.

Aufgabe 1.9 (👉). Gegeben sind Algorithmus A mit Laufzeit $100n^2$ und Algorithmus B mit Laufzeit 2^n . Was ist der kleinste positive Wert von $n \in \mathbb{N}$, sodass Algorithmus A auf demselben Rechner schneller läuft als Algorithmus B?

Aufgabe 1.10 (Hügeligenschaften 🗝️). Sei A ein Array der Länge $n \geq 1$.

- Beweise, dass A immer mindestens einen Hügel hat.
- Was ist die maximale Anzahl an Hügeln, die A haben kann?
- Angenommen wir ändern die Definition von Hügel so, dass $A[i]$ ein Hügel ist, wenn $A[i]$ echt größer als seine Nachbarn ist. Welchen Effekt hat die Änderung auf die beiden Eigenschaften?

Aufgabe 1.11 (Noch mehr Hügel 🗝️).

- Was ist die schlimmste Eingabe für jeden der drei Hügelalgorithmen?
- Schreib den rekursiven Algorithmus so um, dass er iterativ wird. Schreib hierfür den Pseudocode.
- Beweise, dass der rekursive Algorithmus immer einen Hügel findet. *Hinweis:* Definiere eine nützliche Invariante, die für jeden rekursiven Aufruf gilt, und gib einen Beweis per Induktion.
- Implementiere (in einer echten Programmiersprache deiner Wahl) die beiden Linearzeitalgorithmen um Hügel zu finden.
- Implementiere auch den rekursiven Algorithmus, um Hügel zu finden (achte darauf, dass dein Programm immer nur auf die Positionen 0 bis $n - 1$ des Arrays zugreifen darf).

Aufgabe 1.12 (2D-Hügel, 🖍️). Sei M eine $n \times n$ Matrix (ein zweidimensionales Array). Ein Eintrag $M[i][j]$ ist ein Hügel, wenn er nicht kleiner ist als seine Nachbarn im N,O,S,W ist (das heißt $M[i][j] \geq M[i][j-1]$, $M[i][j] \geq M[i+1][j]$, $M[i][j] \geq M[i][j+1]$, und $M[i][j] \geq M[i-1][j]$; die Randfälle sind analog zu den Hügeln im 1D-Fall zu verstehen). Wir wollen nun einen effizienten Algorithmus entwerfen, der einen Hügel in M findet.

- a) 👍 Gib einen einfachen Algorithmus an, der $O(n^2)$ Zeit braucht.
- b) 🔑 Gib einen Algorithmus an, der $O(n \log n)$ Zeit braucht. Begründe, wieso die Laufzeitschranke gilt. *Hinweis:* Beginne, indem du die maximale Zahl in der mittleren Spalte findest, und benutze dies, um das Problem rekursiv zu lösen.
- c) 🐭 Gib einen Algorithmus an, der $O(n)$ Zeit braucht. Begründe, wieso die Laufzeitschranke gilt. *Hinweis:* Konstruiere einen rekursiven Algorithmus, der M in 4 Quadranten aufteilt.

Du brauchst nur einen der drei Aufgabenteile schriftlich abgeben.