Algorithmen & Datenstrukturen

Übungsstunde 7

Michelle Honegger - CAB G59 - 03.11..2024

Miniquiz 6

- Passwort: **programming2**
- 10 Fragen
- 12 Minuten
- 0.1 Punkte pro richtige Antwort

Programm

- Miniquiz
- Rückmeldung Serie 5
- Besprechung Aufgaben
- Theory Recap
- Prüfungsaufgaben
- Common Coding problems

Rückmeldung Serie 5

- 1. Aufgabe sehr gut! Viele schön mit Zwischenschritten
- 2. Aufgabe auch gut
- Bei Beweisen, dass Code richtig ist muss code gereferenced werden
- Auch Base Case muss bewiesen werden

Besprechung Miniquiz

In a linked list data structure L, suppose you have a pointer to an object o in L. Then inserting a key k into the list after o (i.e. $\texttt{insertAfter}(o, k, L)$) takes $O(1)$ time.

- Wahr
- Falsch

Let $B1, B2$ be two leaves in a 2-3-tree. If an insert operation increases the depth of $B1$, then it also increases the depth of $B2$.

- Wahr
- Falsch

We can find the maximum key of a 2-3 tree with n leaves in time $O(\log(n))$.

- Wahr
- Falsch

Let T be a 2-3 tree with depth 4. Let x be the number of leaves. It is possible for x to be equal to:

True	False		
○	○	5	
○	○	10	
○	○	20	
○	○	50	
○	○	100	
○	○	200	
○	○	1000	

Bewertung: **MTF1/0** ?

Consider the subset sum problem with input $A[1 \ldots n]$ and target value b, where the subproblem $T(i, s)$ denotes whether s is a subset sum of $A[1 \ldots i]$. Fill in X such that the recursion is $T(i, s) = T(i-1, s) \vee T(i-1, X)$ (for $2 \leq i \leq n$). If $X < 0$ then $T(i-1, X)$ is considered to be false.

- a. $X = A[i-1]$
- b. $X = s - A[i]$
- c. $X = s - b$
- d. $X = b$

There is a known algorithm which solves subset sum in time $O(n^3)$.

- Wahr
- Falsch

There is an algorithm for knapsack which computes in polynomial time a solution which is at least half as good as the optimal solution.

- Wahr
- Falsch

Consider the knapsack problem with weights w_i, profits p_i and weight limit W. Let $P = p_1 + \ldots + p_n$. Which of the following statements are true/false:

True	False	
○	○	There is an algorithm solving the problem in time $O(nW)$.
○	○	There is an algorithm solving the problem in time $O(nP)$.

Bewertung: **Teilpunkte**

For the knapsack problem, let OPT be the optimal solution for the original profits p_i and $\widetilde{OPT}$ the optimal solution for the rounded profits $\tilde{p}_i$. Which of the following is always true?

- a. $\sum_{i \in \widetilde{OPT}} \tilde{p}_i \geq \sum_{i \in OPT} \tilde{p}_i$.
- b. $\sum_{i \in \widetilde{OPT}} \tilde{p}_i = \sum_{i \in OPT} \tilde{p}_i$.
- c. $\sum_{i \in \widetilde{OPT}} \tilde{p}_i \leq \sum_{i \in OPT} \tilde{p}_i$.
- d. None of the above.

Let L be a longest ascending subsequence of $A[1 \ldots i]$ and L' be a longest ascending subsequence of $A[1 \ldots i+1]$. Then L' is either identical to L, or L' is obtained from L by adding $A[i+1]$ at the end.

- Wahr
- Falsch

Theory Recap

Subset Sum

- Input: Array mit Zahlen und Zahl b (alle in N)
- Restriktion: Ein Subset muss anders wie bei Subarray nicht zusammenhängend sein
- Gesucht: Subset I sodass b die Summe von die Summe der Einträge von A an Stellen i ist

Subset Sum

- Teilproblem: dp[i][j] := Ist j eine Teilsumme von A[1..i]? (boolean)
- Rekursion:

$$dp[i][j] = \begin{cases} 1 & \text{if } j = 0 \\ 0 & \text{if } i = 0 \wedge j \neq 0 \\ dp[i-1][j] \text{ or } dp[i-1, j-A[i]] & \text{, sonst} \end{cases}$$

↖ pseudopolynomiell

$dp[i-1, j-A[i]]$: 0, falls $j < A[i]$ um out of bounds zu vermeiden

	0	1	2	3	4	5	6	7	8
5	1	0	0	0	0	0	0	0	0
5	1	0	0	0	0	1	0	0	0
7	1	0	0	0	0	1	0	1	0
2	1	0	1	0	0	1	0	1	0
4	1	0	1	0	1	1	1	1	0
9	1	0	1	0	1	1	1	1	0
6	1	0	1	0	1	1	1	1	1

1 steht für 'true', 0 für 'false'

Knapsack

- Input: Zahl n als Gewichtslimit und Array der Länge nx2 mit Gewicht und Wert pro Gegenstand
- Gesucht: Subset der Gegenstände mit maximalem Wert unter Gewichtslimit

Knapsack

- Teilproblem: dp[i][j]:= maximaler profit den man aus A[1..i] mit höchst Gewicht w erreichen kann
- Rekursion: dp[i][j] = $max\{dp[i-1][j],\ p_i + dp[i-1][w-w_i]\}$

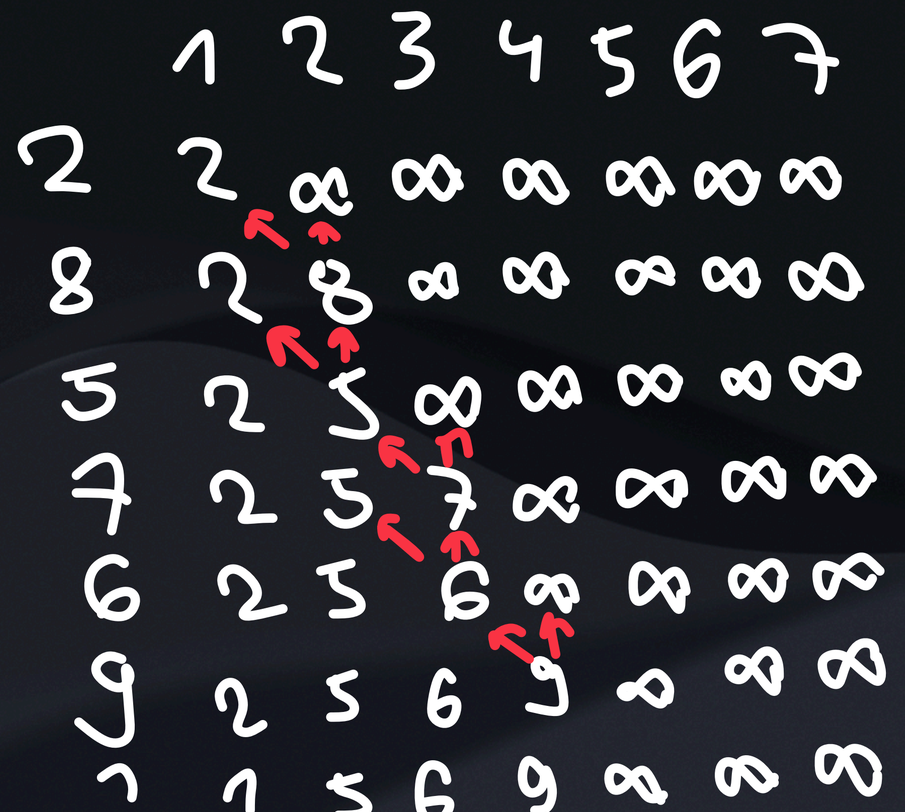
verwende item i nicht

verwende item i schon

- Laufzeit: O(nW) (pseudopolynomiell)

Längste Aufsteigende Teilfolge

- Input: Array der Länge n mit Integers
- Gesucht: Länge der längsten aufsteigenden Teilfolge



Längste aufsteigende Teilfolge

- Teilproblem: dp[i][j]:= kleinstmögliche Endung einer AT der länge j in A[1..i]
- Rekursion: dp[i][j] = $\begin{cases} A[1] & \text{if } i=1 \text{ \& } j=1 \\ \infty & \text{if } i=1 \text{ \& } j>1 \\ A[i] & \text{if } dp[i-1][j-1] < A[i] < dp[i-1,j] \\ dp[i-1][j] & \text{, sonst} \end{cases}$
- Laufzeit: O(nlog(n)) wegen binary search

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$ summing to $A := \sum_{i=1}^{n} a_i$, which is a multiple of 3. You want to determine whether it is possible to partition $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that the corresponding elements of the array yield the same sum, i.e.

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{A}{3}.$$

Note that I, J, K form a partition of $\{1, \ldots, n\}$ if and only if $I \cap J = I \cap K = J \cap K = \emptyset$ and $I \cup J \cup K = \{1, \ldots, n\}$.

For example, the answer for the input $[2, 4, 8, 1, 4, 5, 3]$ is *yes*, because there is the partition $\{3, 4\}$, $\{2, 6\}$, $\{1, 5, 7\}$ (corresponding to the subarrays $[8, 1]$, $[4, 5]$, $[2, 4, 3]$, which are all summing to 9). On the other hand, the answer for the input $[3, 2, 5, 2]$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a partition exists. Your algorithm should have an $\mathcal{O}(nA^2)$ runtime to get full points. Address the following aspects in your solution:

1) *Definition of the DP table*: What are the dimensions of the table $DP[\ldots]$? What is the meaning of each entry?

2) *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.

3) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?

4) *Extracting the solution*: How can the final solution be extracted once the table has been filled?

5) *Running time*: What is the running time of your algorithm? Provide it in Θ-notation in terms of n and A, and justify your answer.

Exercise Recommendations

Aufgabenblatt 7

- 7.1 Bonus
- 7.2 Bonus
- 7.3 Einfach noch mehr DP, gut zum üben, wahrscheinlich habt ihr trotzdem keine Zeit dafür
- 7.4 Bonus
- 7.5 Algorithmisch interessant, aber die am ehesten skippen

Leetcode of the Week

- 72. Edit Distance
- https://leetcode.com/problems/edit-distance/

Peergrading

- Aufgabe 6.3
- Grading Scheme in Lösungen beachten