Algorithms and Datastructures (HS2024)
Week 7

Rui Zhang

November 4, 2024

Contents

1 Revision Theory

This week, you did even more dynamic programming. Instead of going through the individual problems you did, repeating what was already written in the lecture notes, I will give you the definition of the subproblem and a few other pieces of information and from there, you will have to answer the rest of the questions like as if you were doing a DP-Exercise. I hope this will help you in actively recalling content from the lecture.

1.1 Subset Sum

Given an Array $A[1, \ldots, n]$ and b of natural numbers, we want to search for a set of indices I such that all elements in A at those indices sum up to exactly b. One naive approach is to try out all 2^n possible subsets of indices.

To solve this problem more quickly, we introduce the following subproblem / DP-Entry-Definition:

$DP[i][s] := true \Leftrightarrow$ "There exists a subset of $A[1, \ldots, i]$ that sums up to s"

$DP[i][s] := false \Leftrightarrow$ "There exists no subset of $A[1, \ldots, i]$ that sums up to s"

With this, we can also realize that the base cases of this DP-Subproblem is

$DP[i][0] := true$, since we can just take the empty set of indices

$DP[0][s] := false$, for $s \neq 0$ since we have the empty subarray $A[1, \ldots, 0]$

From here, I would like you to solve the rest of the questions for DP yourself as practice. If you cannot find the solution, then you are of course free to check out the lecturen notes or just come to my exercise session and ask me directly. I will explain it to you.

1.2 Knapsack

Imagine you are an evil criminal ETH student stealing the exam solutions for the A&D Exam for this semester. You have a backpack that can hold at most W many sheets of solutions. Now, you have arrived at Professor Lengler's desk. He had been tirelessly working on the solutions. Too bad somebody's gonna steal them now. There are n problems on the exam and each problem g_i has a solution that consists of w_i many sheets of paper. Stealing these solutions would be worth a value of p_i, as this exercise grants p_i many points.

Essentially, you are searching for the subset of indices I representing the exact problems whose solutions you will steal such that the sum $\sum_{i \in I} w_i \leq W$ of all sheets is less than or equal to the amount of sheets that fit in your backpack and $\sum_{i \in I} p_i$ is maximal.

Let us define the following subproblem: $DP[i][w] :=$ "Maximum value reachable by solution sheets that fit into the backpack".

Given this subproblem, explain why the following recursion relation holds and solve all other questions required for a complete DP-Exercise (Base Cases, DP Dimensions, Runtime, etc.):

$$DP[i][w] = \max(DP[i-1][w], p_i + DP[i-1][W - w_i])$$

For $w_i \leq W$, else $DP[i][w] = DP[i-1][w]$.

1.3 Longest Ascending Subsequence

Given an array $A[1, \ldots, n]$ of integers, find the length of the longest ascending subsequence of this array. This subsequence is NOT a subarray, which means that a subsequence does not have to be continuous. To solve this problem, we use the DP-Definition: $DP[i][l]$ is the smallest possible ending of an ascending subsequence of length l in $A[1, \ldots, i]$. Based off of this, answer the classic DP

questions again. At this point, I want to note that you have settled on this solution in the lecture since it had a better runtime than all other DP-attempts you had in the lecture. **The reason for this is that this DP-Definition encoded the most amount of information out of all attempts, which reduced the amount of loops we had to take to calculate each entry and thus the overall runtime.**

2 Exercises

2.1 Exercise Sheet 7 - Priority List

Again, top is most important, bottom is least important.

1. 7.2, 7.5 - Good DP Exercises that could appear in exams
2. 7.1 - Still a bonus question, so could also be exam relevant
3. 7.4 - Clarifies the subtle (!) brute-force character of dynamic programming
4. 7.3 - unlikely to be an exam DP exercise judging based on recent exams. Do not quote me on this though. Anything can happen at ETH ;)

2.2 Exercise Sheet 5 - Feedback

1. Many of you misunderstood the heapify task and the definition of a level. The idea is that the bottom-most layer is $height(T)$ and the top-most layer is 0. This means that for this heapify algorithm, you would have had to do the induction step for $t \to t-1$, not $t \to t+1$.
2. Additionally, in the same task, many of you failed to mention that the heapify algorithm, after swapping the parent and a child, follows the parent and continues swapping until the node in quesion reaches a point in which the heap-condition is fulfilled. This procedure is essential and it happens in the innermost loop.
3. In exercise 5.3), many of you also did induction. It is awesome to see so much love towards induction, but in my opinion, induction feels a bit overkill in this case, especially since it does not become that clear where the induction hypothesis should be used in a sensical manner in the induction step. A simple explanation elaborating upon the individual summands of the recursive formula by pointing to the pseudocode with valid arguments is enough.

3 Supplementary Exercises

Dynamic Programming Exercises are often to be found on Leet Code. Go check that website out for more material.

3.1 HS21 T3 - Double Subsetsum

This is another exercise from an old exam. You can find the solution on VIS Community Solutions.

Given n natural numbers $a_1, a_2, a_3, \ldots, a_n$ and $A, B \in \mathbb{N}$, determine if there is a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = A$ and $\sum_{i \in I} a_i^2 = B$. The runtime should be at most $O(n \cdot A \cdot B)$

3.2 Total Denomination of Coins

Check out the following link of a dynamic programming exercise that I find especially interesting and that is similar to the one we did in class last week (see notes week 6 - coins) Total Denomination of Coins