Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, David Steurer
Kasper Lindberg, Lucas Slot, Hongjie Chen, Manuel Wiedmer

4 November 2024

Algorithms & Data Structures — Exercise sheet 7 — HS 24

The solutions for this sheet are submitted on Moodle until 10 November 2024, 23:59.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Exercise 7.1 *Subset sums with duplicates* **(1 point)**.

Let $A[1 \dots n]$ be an array containing n positive integers and let $b \in \mathbb{N}$. We want to know if we can write b as a subset sum of A where each element of A is allowed to be used once, twice or not at all. If this is possible, we say b is a subset sum of A with duplicates.

For example, consider $A = [3, 4, 2, 7]$ and $b = 22$. Then b is a subset sum of A with duplicates, as we can use 3 not at all, use 4 once, and use 2 and 7 twice. In other words, we can write b as $0 \cdot A[1] + 1 \cdot A[2] + 2 \cdot A[3] + 2 \cdot A[4]$.

Describe a DP algorithm that, given an array $A[1 \dots n]$ of positive integers and a positive integer b, returns True if and only if b is a subset sum of A with duplicates. Your algorithm should have asymptotic runtime complexity at most $O(b \cdot n)$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Exercise 7.2 *Road trip.*

You are planning a road trip for your summer holidays. You want to start from city C_0, and follow the only road that goes to city C_n from there. On this road from C_0 to C_n, there are $n - 1$ other cities $C_1, \dots, C_{n-1}$ that you would be interested in visiting (all cities $C_1, \dots, C_{n-1}$ are on the road from C_0 to C_n). For each $0 \leq i \leq n$, the city C_i is at kilometer k_i of the road for some given $0 = k_0 < k_1 < \dots < k_{n-1} < k_n$.

You want to decide in which cities among $C_1, \ldots, C_{n-1}$ you will make an additional stop (you will stop in C_0 and C_n anyway). However, you do not want to drive more than d kilometers without making a stop in some city, for some given value $d > 0$ (we assume that $k_i < k_{i-1} + d$ for all $i \in [n]$ so that this is satisfiable), and you also don't want to travel backwards (so from some city C_i you can only go forward to cities C_j with $j > i$).

(a) Provide a *dynamic programming* algorithm that computes the number of possible routes from C_0 to C_n that satisfy these conditions, i.e., the number of allowed subsets of stop-cities. Your algorithm should have $O(n^2)$ runtime.

In your solution, address the following aspects:

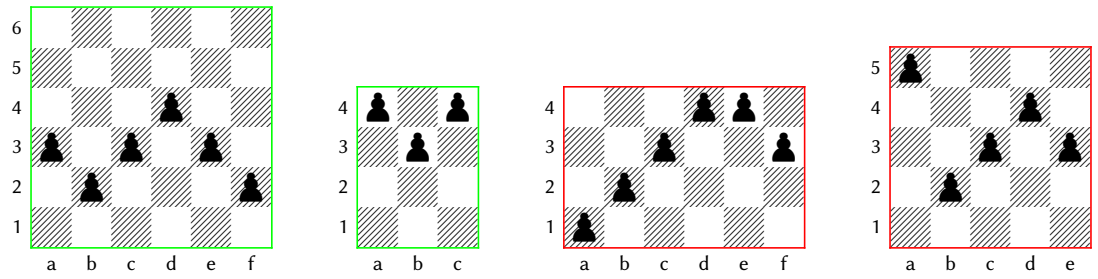
1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

(b) If you know that $k_i > k_{i-1} + d/10$ for every $i \in [n]$, how can you turn the above algorithm into a linear time algorithm (i.e., an algorithm that has $O(n)$ runtime) ?

Exercise 7.3 *Safe pawn lines.*

On an $N \times M$ chessboard (N being the number of rows and M the number of columns), a *safe pawn line* is a set of M pawns with exactly one pawn per column of the chessboard, and such that every two pawns from adjacent columns are located diagonally to each other. When a pawn line is not safe, it is called *unsafe*.

The first two chessboards below show safe pawn lines, the latter two unsafe ones. The line on the third chessboard is unsafe because pawns d4 and e4 are located on the same row (rather than diagonally); the line on the fourth chessboard is unsafe because pawn a5 has no diagonal neighbor at all.



Describe a DP algorithm that, given $N, M > 0$, counts the number of safe pawn lines on an $N \times M$ chessboard. Your solution should have complexity at most $O(NM)$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Exercise 7.4 *Weight and volume knapsack* **(1 point)**.

Consider a knapsack problem with n items with profits being positive integers in the array $P[1, \ldots, n]$, and weights being positive integer in the array $W[1, \ldots, n]$. The knapsack has a weight limit $W_{max} \in \mathbb{N}$. Furthermore, each item has a volume of 1 and the knapsack has a volume limit V_{max}.

Describe a DP algorithm that, given the arrays $P[1, \ldots, n]$, $W[1, \ldots, n]$, of positive integers and positive integers W_{max}, V_{max}, returns the total profit of the largest subset of items such that the items respect both the weight limit and volume limit of W_{max} and V_{max}. Your algorithm should have asymptotic runtime complexity at most $O(n \cdot W_{max} \cdot V_{max})$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Exercise 7.5 *Zebra arrays* **(1 point)**.

A square two-dimensional array $Z[1 \ldots k][1 \ldots k]$ with entries in $\{0, 1\}$ is called a *zebra array* if no two adjacent entries of Z are equal. We say two distinct entries $Z[i_1][j_1]$ and $Z[i_2][j_2]$ in Z are adjacent if

- $i_1 = i_2$ and $|j_1 - j_2| \leq 1$; or
- $|i_1 - i_2| \leq 1$ and $j_1 = j_2$.

Describe a DP algorithm that, given a two-dimensional array $A[1 \ldots n][1 \ldots m]$ with entries in $\{0, 1\}$, outputs the size of a largest zebra array contained in A. That is, the largest k such that, for some $1 \leq i \leq n - k + 1, 1 \leq j \leq m - k + 1$, the array $A[i \ldots i + k - 1][j \ldots j + k - 1]$ is a zebra array. Your algorithm should have asymptotic runtime complexity at most $O(nm)$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Hint: *U*se a DP table $B[1 \dots n][1 \dots m]$. The meaning of entry $B[i][j]$ is

$$B[i][j] = \text{size of the largest zebra array in } A \text{ whose bottom-right entry is } A[i][j].$$

Hint: *Y*our recursion to compute $B[i][j]$ should involve the entries $B[i][j-1]$, $B[i-1][j]$, $B[i-1][j-1]$ and also the entries $A[i][j]$, $A[i][j-1]$, $A[i-1][j]$, $A[i-1][j-1]$.