Serie 7

Exercise 7.1

Let $A[1,\dots,n]$ be an array containing n positive integers, and let $b \in \mathbb{N}$. We want to know if there exists a subset $I \subseteq \{1,2,\dots,n\}$, together with multipliers $c_i \in \{1,3\}$, $i \in I$ such that:

$$b = \sum_{i\in I} c_i \cdot A[i].$$

If this is possible, we say b is a 1-3 subset sum of A. For example, if $A = [16,4,2,7,11,1]$ and $b = 61$, we could write $b = 3\cdot 16 + 4 + 3\cdot 2 + 3\cdot 1$.

Describe a DP algorithm that, given an array $A[1,\dots,n]$ of positive integers, and a positive integer $b \in \mathbb{N}$ returns True if and only if b is a 1-3 subset sum of A. Your algorithm should have asymptotic runtime complexity at most $O(b\cdot n)$.

Dimensions of DP:

$(n+1) \times (b+1)$

Subproblems

DP[i][j] : True if j is a Subset sum of A[1, ⋯ , i]

Recursion

$DP[i][j] = DP[i-1][j] \vee DP[i-1][j-A[i]] \vee DP[i-1][j-3\cdot A[i]]$ Für $i \geq 1$.

B.C. : Out of bounds = False

$DP[i][0] = \text{True}$ $\forall 0 \le i \le n$, $DP[0][j] = \text{False}$ if $j \geq 1$

Calc. order: Increasing i, and then increasing j   Extract sol: DP[n][b]   Runtime: $O(nb)$

Exercise 7.4

Given a binary string $S \in \{0,1\}^n$ of length n, let $f(S)$ be the number of times "11" occurs in the string, i.e. the number of times a 1 is followed by another 1. In particular, the occurrences do not need to be disjoint. For example $f(\text{"}\underline{1}\underline{1}10\underline{1}1\text{"}) = 3$ because the string contains three 1 that are followed by another 1 (underlined). Given n and k, the goal is to count the number of binary strings S of length n with $f(S) = k$.

Describe a DP algorithm that, given positive integers n and k with $k < n$, reports the required number. In your solution, address the same six aspects as in Exercise 7.1. Your solution should have complexity at most $O(nk)$.

Hint: *Define a three dimensional DP table $DP[1\dots n][0\dots k][0\dots 1]$.*

Hint: *The entry $DP[i][j][l]$ counts the number of strings of length i with j occurrences of "11" that end in l (where $1 \le i \le n$, $0 \le j \le k$ and $0 \le l \le 1$).*

Dimensions:

$(n+1) \times (k+1) \times 2$

Subproblem:

DP[i][j][l] := # Strings der Länge i mit j "11" Substrings, die auf l enden

Recursion: <u>B.C.</u>

$DP[0][j][l] = 0$ $0 \le j \le n$ $0 \le l \le 1$

$DP[1][0][l] = 1$ $\leftarrow \underline{0}, \underline{1}$

$l = 0$

$DP[i][j][0] = DP[i-1][j][0] + DP[i-1][j][1]$ $-----1\underline{0}$

$$DP[i][j][1] = \begin{cases} DP[i-1][j][0] + DP[i-1][j-1][1] & \text{if } j>0 \\ DP[i-1][j][0] \end{cases}$$ $-----1\underline{1}$

Calculation order: increasing i, increasing j and l.

Extracting the solution: DP[n][k][0] + DP[n][k][1]

Runtime: $O(nk)$

Exercise 7.5

Consider a knapsack problem with n items with values $v_i \in \mathbb{N}$ and weights $w_i \in \mathbb{N}$ for $i \in \{1, 2, \dots, n\}$, and weight limit $W \in \mathbb{N}$. Assume[1] that $w_{\max} := \max_{1 \le i \le n} w_i \le W$ and also that $W \le \sum_{i=1}^n w_i$.

For a set of items $I \subseteq \{1, 2, \dots, n\}$, we write $v(I) = \sum_{i \in I} v_i$ and $w(I) = \sum_{i \in I} w_i$ for the total value (resp. weight) of I. So, the solution to a knapsack problem is given by

$$\text{opt} := \max \{v(I) : I \subseteq \{1, 2, \dots, n\}, \quad w(I) \le W\}.$$

Let $\varepsilon > 0$. We say a set of items $I \subseteq \{1, 2, \dots, n\}$ is ε-*feasible* if it violates the weight limit by at most a factor $1 + \varepsilon$, that is, if

$$w(I) \le (1 + \varepsilon) \cdot W.$$

In this exercise, we construct an algorithm that finds an ε-feasible set I with $v(I) \ge \text{opt}$ in polynomial time in n and $1/\varepsilon$.

(a) Let $k \in \mathbb{N}$. Consider a 'rounded version' of the knapsack problem above obtained by replacing the weights w_i by:

$$\widetilde{w_i} := k \cdot \left\lfloor \frac{w_i}{k} \right\rfloor.$$

Recall the dynamical programming algorithm you have seen in the lecture which solves the knapsack problem in time $O(n \cdot W)$. Explain how to modify this algorithm to solve the 'rounded version' in time $O((W/k) \cdot n)$. For simplicity, you may assume that W is an integer multiple of k for this part.

Hint: *After rounding, all the $\widetilde{w_i}$ are integer multiples of k. Use this to reduce the number of table entries you have to compute in the dynamical programming algorithm.*

We write opt_k for the optimum solution value of the rounded problem in part (a). From now on, you may assume that your modified algorithm also returns a set I_k with $v(I_k) = \text{opt}_k$ and $\sum_{i \in I_k} \widetilde{w_i} \le W$.

(b) Explain why $\text{opt}_k \ge \text{opt}$.

$$\widetilde{w}_i \le w_i$$

$$\forall\ I \quad w(I) \le W \quad \sum_{i \in I} \widetilde{w}_i \le W$$

$$\Rightarrow \text{opt}_k \ge \text{opt}$$

(c) Set $\varepsilon := (nk)/w_{\max}$. Show that $w(I_k) \le (1 + \varepsilon) \cdot W$, that is, show that I_k is ε-feasible.

Hint: *Show that $w_i \le \widetilde{w_i} + k$ for each $i \in \{1, 2, \dots, n\}$. We know that $\sum_{i \in I_k} \widetilde{w_i} \le W$. Combine these facts to show that $w(I_k) := \sum_{i \in I_k} w_i \le W + n \cdot k$. Finally, use the fact that $W/w_{\max} \ge 1$ to conclude your proof.*

$$\lfloor w_i/k \rfloor \ge \frac{w_i}{k} - 1 \quad \Rightarrow \quad \underbrace{k \cdot \lfloor w_i/k \rfloor}_{= \widetilde{w}_i} \ge k\left(\frac{w_i}{k} - 1\right) \quad \Rightarrow \quad \widetilde{w}_i \ge k \cdot \left(\frac{w_i}{k} - 1\right) = w_i - k \quad \forall i \in \{1, \dots, n\}$$

$$w_{\max} = \max_{1 \le i \le n} w_i \le W$$

$$w(I_k) = \sum_{i \in I_k} w_i \le \sum_{i \in I_k} (\widetilde{w}_i + k) = \sum_{i \in I_k} \widetilde{w}_i + \sum_{i \in I_k} k \le W + n \cdot k \le W + \frac{W}{w_{\max}} (nk) = W\Big(1 + \underbrace{\frac{nk}{w_{\max}}}_{\varepsilon}\Big)$$

(d) Now let $\varepsilon > 0$ be arbitrary. Describe an algorithm that finds an ε-feasible set I of items with $v(I) \ge \text{opt}$ in time $O(n^3/\varepsilon)$. Prove the runtime guarantee and correctness of your algorithm.

Hint: *Apply the algorithm of part (a) to the rounded problem with $k = (w_{\max} \cdot \varepsilon)/n$. For simplicity, you may assume that this k is an integer in your proof. Then, use the assumption that $W \le \sum_{i=1}^n w_i$ $(\le n \cdot w_{\max})$ to bound the runtime of the algorithm in terms of n and $1/\varepsilon$. Finally, use part (b) and part (c) to show correctness.*

$$O(n^3/\varepsilon)$$

$$(W/k) \cdot n = \frac{n \cdot W}{w_{\max} \cdot \varepsilon} \cdot n = \frac{n^2}{\varepsilon} \cdot \frac{W}{w_{\max}} \le \frac{n^2}{\varepsilon} \cdot \frac{n \cdot w_{\max}}{w_{\max}} \le \frac{n^3}{\varepsilon} \,/\!/$$

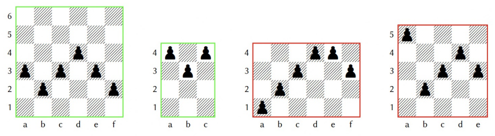
$$I_k \qquad v(I_k) \ge \text{opt} \qquad k = (w_{\max} \cdot \varepsilon)/n \quad \Rightarrow \quad \varepsilon = \frac{nk}{w_{\max}} \qquad w(I_k) \le (1+\varepsilon) \cdot W$$

Exercise 7.3

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*.

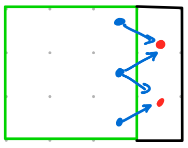
The first two chessboards below show safe pawn lines, the latter two unsafe ones. The line on the third chessboard is unsafe because pawns d4 and e4 are located on the same row (rather than diagonally); the line on the fourth chessboard is unsafe because pawn a5 has no diagonal neighbor at all.



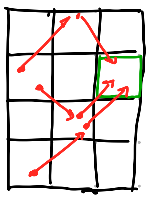
Describe a DP algorithm that, given $N, M > 0$, counts the number of safe pawn lines on an $N \times M$ chessboard. In your solution, address the same six aspects as in Exercise 7.1. Your solution should have complexity at most $O(NM)$.

Dimensions: DP[1… N][1… M]

Meaning of an entry: DP[i][j] := # Untersch. safe pawn lines auf einem $N \times j$ board, die in Zeile i enden.



$N = 4$ DP[2][3] = 3



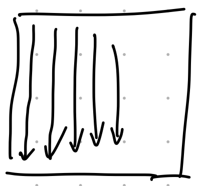
Base Case: DP[i][1] = 1

Edge Cases: DP[1][j] = DP[2][j-1]

DP[N][j] = DP[N-1][j-1]

Computation: DP[i][j] = DP[i-1][j-1] + DP[i+1][j-1]

Calculation order: Increasing j, then increasing i



Extracting the solution:

$$\sum_{i=1}^{N} DP[i][M]$$

Runtime:

$O(MN)$