Exercises for Algorithms and Data Structures Lectures 8 and 9

Question 1. (old exam question)

Along a long stretch of highway are $n \geq 1$ positions, which all lie 500 meters apart, these are designated for billboards to be placed. The (expected) yield from such a billboard depends on the position where it stands and is given by a 1-dimensional array: yield$[i]$ (> 0; $i = 1, \ldots, n$) = yield from a billboard on spot i. Billboards should always be standing more than ($>$) 1 km from each other. The intention is to position billboards in such a way that the maximum total yield is obtained.

- **a.** What is the maximum total yield if $n = 1$? And if $n = 2$? And if $n = 3$?
- **b.** Give a recursive formula for `maxtotal`(n), the maximum total yield. The base cases ($n = 1, 2, 3$) were calculated in **a**.
- **c.** The problem can be solved recursively, but that is in this case not efficient. Explain why not, and indicate how dynamic programming can considerably improve the efficiency. Then discuss (briefly) both the top-down method as the bottom-up method for dynamic programming and explain the difference between these two methods.
- **d.** Provide a bottom-up dynamic programming algorithm that calculates the maximum total yield, either in pseudocode or in Python code. Clearly formulate what kind of array you are using and give the recurrence relation according to which the array is filled.

Question 2. (Levitin, exercise 8.1.12)

World Series odds

Consider two teams A and B playing a series of games until one of the teams has won n games. Assume that the probability of A winning a game is the same in every game, i.e. p. There are no ties, so the probability of A losing a game is $q = 1 - p$. Let $P(i, j)$ be the probability that team A wins the series of games, when A still needs i games to reach n and B still needs j games.

- **a.** Give a recurrence relation for $P(i, j)$ that can be used by a Dynamic Programming algorithm.
- **b.** Determine the probability that A wins the series of games, when there are four games already won (ie $n = 4$) and the probability that A wins a game is 0.4 (ie $p = 0.4$).
- **c.** Give pseudocode for a DP algorithm that solves this problem, and determine its time and space complexity.

Question 3.

We have been given $n \geq 1$ different weights $g_1, g_2, \ldots, g_n$ and an integer $w \geq 0$. All of these variables are positive integers (> 0). Let all of these weights be saved in an array called `weights`. Is it possible to pick a subset of weights such that the sum of the weights is exactly equal to x?

- **a.** Write an exhaustive recursive function (returning a boolean) that solves this problem by returning True (it is possible) or False (it is not possible).

b. Now solve the same problem with bottom-up dynamic programming. To do this, derive a recurrence equation `weighing`(i,j) from the recursive function that you have written above. Hint: it is useful to see `weighing`(i,j) as a boolean array `weighing`[i][j] (also see the next subquestion).

c. Think about the order of computation that would work to implement this bottom-up dynamic programming algorithm, and write the algorithm in Python code. The value we are interested in at the end is `weighing`[n][w].

d. Explain how we can derive the solution from the boolean array `weighing` (solution = the subset of weights that we used to obtain the desired sum of weights).

e. If we are solely interested in the boolean outcome (true/false), we can also use a 1D array `weighing`. Explain how the algorithm in **c.** should be adjusted for this.

Question 4.

We look at the Japanese game Pachinko, played on a special machine (a kind of pinball machine), which we model as follows.
We have an upright grid with height m and width n. At the top you can throw a ball into a column, which then falls perpendicularly down until it hits an obstacle (a *) or until it ends up in a gate (= a cell with a `number` in it), or until it falls off the grid at the bottom. In the latter case you have earned nothing. If the ball lands in a gate, you're done and you've won `number` euros. If the ball hits a * along the way, it will fall further down the column to the left or right of the *: the probability that it falls to the left is equal to the probability that it falls to the right (so both probability 0.5).
Assumption: the grid does not contain any obstacles directly next to each other (nor in the same row, neither in the same column nor on the same diagonal). Same for the gates. Furthermore, the first and last column contain no obstacles or gates.
The goal is to find the maximum expected yield and the column in which you have to throw the ball to achieve this.

Example: the maximum expected yield for the grid below is 7,50 euros.

```
. . . 1 . . . .
. . . . . . . .
. . * . . . * .
. . . . * . . .
. 1 . . . . . .
. . . * . * . .
. . . . . . . .
. . 9 . 7 . 7 .
```

a. Give a recursive algorithm that calculates the maximum expected yield.

b. Give an example that shows that using recursion alone to solve this problem is generally inefficient.

c. Use bottom up dynamic programming to calculate the maximum expected yield. So: choose a suitable (two-dimensional) array D, formulate a recurrence relation that indicates how $D[i][j]$ is calculated, specify a calculation order and write an algorithm that fills the array. Also indicate how to find the column corresponding to the maximum expected yield.

d. Do you think it is more convenient here to use top down dynamic programming (= recursion with array) than bottom up or not? Motivate your answer.

Question 5. (exam question june 2014)

Jan recently bought a holiday home in the mountains. He likes walking and regularly invites people for a walk through the area. He has a global map of the area with the average heights per hectare. Because Jan doesn't know the area very well yet and doesn't want to get lost, he decides to just walk due south and due east from hectare to hectare on the map. He walks from the top left corner of the map, where his house is located, to the bottom right corner, where he treats his guests to a meal. The top left corner is at position (row, column) = $(0, 0)$, the bottom right corner is at $(n-1, n-1)$.

Example: a height map $H[i][j]$ for $n = 6$:

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

The height difference between two adjacent hectares is defined as the absolute difference between the two heights of these hectares and is therefore always ≥ 0.
The height difference of a walking route is equal to the sum of the height differences on this route. The height difference of the walking route (3.0), (3.1), (3.2), ..., (3.5) is $6+2+3+1+2 = 14$.

a. For his grandmother, Jan is looking for the route with as little height difference as possible. Describe a *greedy* algorithm for finding such a route. Which route does he find when he applies this algorithm to the example above? What is the total height difference?

Jan wants to solve the problem with bottom-up dynamic programming. He uses a two-dimensional array M, where $M[i][j]$ is the minimum height difference for a walking route from (i, j) to $(n-1, n-1)$. We eventually want to know $M[0][0]$.

b. Explain why M for $0 \leq i < n$ and $0 \leq j < n$ satisfies:

$$M[i][j] = \begin{cases} 0 & \text{if } i = n-1 \text{ and } j = n-1 \\ M[n-1][j+1] + \mathbf{abs}(H[n-1][j+1] - H[n-1][j]) & \text{if } i = n-1 \text{ and } 0 \leq j < n-1 \\ M[i+1][n-1] + \mathbf{abs}(H[i+1][n-1] - H[i][n-1]) & \text{if } j = n-1 \text{ and } 0 \leq i < n-1 \\ \mathbf{min}(M[i+1][j] + \mathbf{abs}(H[i+1][j] - H[i][j]), \\ \quad M[i][j+1] + \mathbf{abs}(H[i][j+1] - H[i][j])) & \text{if } 0 \leq i < n-1 \text{ and } 0 \leq j < n-1 \end{cases}$$

where $\mathbf{abs}(a)$ is the absolute value of a and $\mathbf{min}(a,b)$ returns the minimum of a and b.

c. Write an algorithm in pseudocode or Python code that, using the array M and the recurrence relation from question **b.**, computes the requested minimum height difference. You can assume that the functions **min** and **abs** already exist.
State clearly in which order you fill the array M, and explain why in that particular order.

Applying the algorithm from question **c.** to the example, we get the following array M:

12	11	9	8	7	6
11	10	10	10	7	5
15	8	7	6	6	4
15	11	9	6	5	3
11	10	9	7	6	2
14	13	10	6	5	0

d. From this M, determine the minimum height difference and a corresponding optimal walking route from (0, 0) to $(n-1, n-1)$. Explain, in words, how you determined this walking route from the arrays M and H.