Eidgenössische
Technische Hochschule
Zürich

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

Departement of Computer Science
Markus Püschel, David Steurer

19. November 2018

Algorithms & Data Structures — Homework 9 — HS 18

Exercise Class (Room & TA): ______________________

Submitted by: ______________________

Peer Feedback by: ______________________

Points: ____

Hint: This exercise sheet is concerned with *dynamic programming*. A complete description of a dynamic program always includes the following aspects (important also for the exam!):

1. **Definition of the DP table:** What are the dimensions of the dynamic programming table DP[., .]? What is the meaning of each entry (in clearly worded words)?
2. **Calculation of an entry:** Which values of the table are initialized, and how are they initialized? How are entries calculated from other entries? What are the dependencies between entries?
3. **Calculation order:** In what order can you calculate the entries so that these dependencies are fulfilled?
4. **Reading the solution:** How can the solution be read out from the table at the end?

Exercise 9.1 *Longest Ascending Subsequence.*

The longest ascending subsequence problem is concerned with finding a longest subsequence of a given array A of length n such that the subsequence is sorted in ascending order. The subsequence does not have to be contiguous and it may not be unique. For example if $A = [1, 5, 4, 2, 8]$, a longest ascending subsequence is $1, 5, 8$. Other solutions are $1, 4, 8$, and $1, 2, 8$.

Given is the array:

$$[19, 3, 7, 1, 4, 15, 18, 16, 14, 6, 5, 10, 12, 19, 13, 17, 20, 8, 14, 11]$$

Use the dynamic programming algorithm as described in class or the script to find the length of a longest ascending subsequence and the subsequence itself. Show all necessary tables and information you used to obtain the solution.

Solution: We represent the solution by two tables, DP_1 and DP_2. $DP_1[i]$ contains the length of the longest subsequence ending at index i, and $DP_2[i]$ contains the index of the previous entry in the subsequence.

$DP_1 =$	1	1	2	1	2	3	4	4	3	3	3	4	5	6	6	7	8	4	7	5

$DP_2 =$	0	0	2	0	4	5	6	6	5	5	5	10	11	12	12	14	15	10	14	11

The longest subsequence can be found at index 16. From $DP_1[16]$ and $DP_2[16]$, the longest ascending subsequence is $1, 4, 6, 10, 12, 13, 17, 20$, and it has length 8.

Exercise 9.2 *Longest Common Subsequence.*

Given are two arrays, A of length n, and B of length m, we want to find the their longest common subsequence and its length. The subsequence does not have to be contiguous. For example, if $A = [1, 8, 5, 2, 3, 4]$ and $B = [8, 2, 5, 1, 9, 3]$, a longest common subsequence is $8, 5, 3$ and its length is 3. Notice that $8, 2, 3$ is another longest common subsequence.

Given are the two arrays:

$$A = [7, 6, 3, 2, 8, 4, 5, 1]$$

and

$$B = [3, 9, 10, 8, 7, 1, 2, 6, 4, 5],$$

Use the dynamic programming algorithm as described in class or the script to find the length of a longest common subsequence and the subsequence itself. Show all necessary tables and information you used to obtain the solution.

Solution: DP table:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	0	0	1	1	1	2	2	2
3	0	1	1	1	1	1	1	1	2	2	2
4	0	1	1	1	1	1	1	2	2	2	2
5	0	1	1	1	2	2	2	2	2	2	2
6	0	1	1	1	2	2	2	2	2	3	3
7	0	1	1	1	2	2	2	2	2	3	4
8	0	1	1	1	2	2	3	3	3	3	4

To find some longest common subsequence, we create an array S of length $DP[n, m]$ and then we start moving from cell (n, m) of the DP table in the following way:

If we are in cell (i, j) and $DP[i-1, j] = DP[i, j]$, we move to $DP[i-1, j]$.

Otherwise, if $DP[i, j-1] = DP[i, j]$, we move to $DP[i, j-1]$.

Otherwise, by definition of DP table, $DP[i-1, j-1] = DP[i, j] - 1$ and $A[i-1] = B[j-1]$, so we assign $S[DP[i-1, j-1]] \leftarrow A[i-1]$ and then we move to $DP[i-1, j-1]$.

We stop when $i = 0$ or $j = 0$.

Using this procedure we find the following longest common subsequence: $S = [7, 6, 4, 5]$.

Exercise 9.3 *Gym Schedule* **(1 Point)**.

Alice likes to lift weights at the gym and wants to schedule her gym going for the next N days. Each day, Alice either goes to the gym or not, so a schedule can be thought of as a list of length N where the

i^{th} entry contains whether or not Alice goes to the gym that day. Alice cannot go to the gym on two consecutive days, or else she will become too fatigued.

Use dynamic programming to help Alice calculate the number of different gym schedules under this constraint. Note that one valid schedule is that Alice will not go to the gym at all during those N days.

Solution:

Definition of the DP table: DP is an array of length n (indexed starting with 1). $DP[i]$ contains the number of possible shedules if there are i days.

Calculation of an entry: Initialize $DP[1]$ to 2: Alice can either go to the gym or not on day 1. Initialize $DP[2]$ to 3: Alice can go on the first day or go on the second day or go on neither day.

An entry $i > 2$ can be calculated as follows: $DP[i]$ can be calculated by adding up the number of possible shedules if Alice goes to the gym on day i plus the number of possible shedules if she does not go on day i.

- If Alice does not go to the gym on day i, this places no restriction on the shedules, so the number of possible shedules in this event is $D[i-1]$.
- If Alice does go to the gym on day i, she can't go to the gym on day $i-1$. We've placed a restriction on day $i-1$ and i but not on any days before that, so the number of possible schedules in this case is equal to $D[i-2]$.

Then, $DP[i] = DP[i-1] + DP[i-2]$.

Calculation order: We can calculate the entries of DP from smallest to largest.

Reading the solution: All we have to do is read the value at $DP[N]$.

Exercise 9.4 *Black and White Stones* **(2 Points)**.

Two friends named Tim and Gordon play a game. They takes turns drawing stones from a bag. The bag contains black stones and white stones. Whoever draws the final black stone wins the game. The bag is opaque and the stones are indistinguishable by touch—thus, Tim and Gordon draw stones from the bag randomly without knowing their color in advance. Tim always draws first. The bag is guaranteed to contain at least one black stone.

Describe a dynamic program to determine the probability that Tim will win the game, if it is played with m black stones and n white stones.

Solution:

Definition of the DP table: Let DP be a dynamic programming table of size $m+1 \times n+1$. When $i \geq 1$, $DP[i,j]$ contains the probability that the first person who draws a stone will win the game if the bag initially contains i black stones and j white ones.

There will always be at least one black pebble to start the game, and in such a case, it is not defined who will win, so the first row of DP is not meaningful. Nevertheless, we will use this row in the DP table to calculate other entries.

Calculation of an entry:

We initialize DP as follows: If there are 0 white pebbles, the last person who draws a pebble wins. Thus set $D[i, 0]$ to 0 if i is even, and otherwise set it to 1. The first row, $DP[0, .]$ is not meaningful, but initialize it 0.

Suppose that someone, Person A, is about to pull a stone from the bag with i black stones and j white stones. Then:

1. With probability $\frac{i}{i+j}$, Person A draws a black stone. If $i = 1$, Person A wins. If $i > 1$, we play a "subgame" with $i-1$ black stones and n white stones, except with their adversary, Person B, drawing first. The probability that Person B wins the subgame is stored in entry $DP[i-1, j]$. In either case, if Person A draws a black stone, they will win with probability $1 - DP[i-1, j]$. When $i = 1$, this is true because we initialized the first row of DP to zero.

   The probability that Person A pulls a black stone and then proceeds to win the game can be obtained by: $\frac{i}{i+j}(1 - DP[i-1, j])$.

2. With probability $\frac{j}{i+j}$, Person A draws a white stone. In this case, we play a subgame with i black stones and $j-1$ white stones, with Person B drawing first. The probability that Person B wins the ensuing subgame is $1 - DP[i, j-1]$,

   The probability that Person A pulls a white stone and then proceeds to win the game is $\frac{j}{i+j}(1 - DP[i, j-1])$.

Each entry depends on the entry above it, and the entry to the left of it.

Calculation order: We can calculate the entries of DP row-by-row from top to bottom, and then within each row from left to right.

Reading the solution: $DP[m, n]$ contains the probability that Tim wins the game.

Then, use the following dynamic programming rule to find $P(i, j)$:

When $m = 5$ and $n = 5$, DP is:

0	0	0	0	0	0
1	.5	.667	.5	.6	.5
0	.667	.333	.6	.4	.571
1	.25	.7	.35	.629	.393
0	.8	.267	.686	.343	.635
1	.166	.762	.286	.683	.341

The probability of Tim winning is .341.

Submission: On Monday, 26.11.2018, hand in your solution to your TA *before* the exercise class starts.