Plan

- Preview Assignment 7
- Dynamic Programming problem solving
  - ↳ HS22, T3
  - ↳ FS21, T3
- Exercise Templates are uploaded (QR-code)

New Groups

Siro Aebersold	saebersold@student.ethz.ch	1
Yathusan Anpalagan	yanpalagan@student.ethz.ch	7
Benedikt Baumgarten	bbaumgarten@student.ethz.ch	2
Beatrice Bold	bebold@student.ethz.ch	5
Yilmaz Bozkan	ybozkan@student.ethz.ch	16
Thomas Bundi	tbundi@student.ethz.ch	8
Cara Cerny	ccerny@student.ethz.ch	4
Adrian Efford Fernandez	aefford@student.ethz.ch	10
Ehad Evgin	eevgin@student.ethz.ch	14
Franco Fierro Brandes	ffierro@student.ethz.ch	15
Fabio Filipponi	ffilipponi@student.ethz.ch	9
Cyril Gremaud	cgremaud@student.ethz.ch	12
Julian Isser	jisser@student.ethz.ch	1
Maximilian Käfer	mkaefer@student.ethz.ch	6
Ashley Kellenberger	akellenberge@student.ethz.ch	16
Alex Lanz	lanzal@student.ethz.ch	7
Ruilai Li	ruilli@student.ethz.ch	13
Anna Limpaset	alimpaset@student.ethz.ch	3
Hafiz Ullah Mija Khel	hmijakhel@student.ethz.ch	10
Yannick Niederer	yniederer@student.ethz.ch	8
Nikol Nikolova	nnikolova@student.ethz.ch	3
Jannis Piekarek	jpiekarek@student.ethz.ch	14
Kiyan Rassouli	krassouli@student.ethz.ch	11
Lars Schell	laschell@student.ethz.ch	5
Camil Schmid	camschmid@student.ethz.ch	9
Dominik Schwaiger	dschwaiger@student.ethz.ch	15
Raoul Sidler	sidlerr@student.ethz.ch	2
Jonathan Soemer	jsoemer@student.ethz.ch	13
Kristof Szadeczky-Kardoss	kszadeczky@student.ethz.ch	12
Vinzenz Wolf	wolfvi@student.ethz.ch	11
Helena Yan	helyan@student.ethz.ch	4
Guohong Zhang	zhanggu@student.ethz.ch	6

Corrections Assignments 5&6

- both will come this week! (sorry for delay)
- Assignment 6 has rather harsh grading (by order of the Head-TA)

Preview Assignment 7

- Only DP ☺
- I can recommend exercise 7.2 & 7.3 (3-dimensional DP for 7.3 → good practice), even if they're not bonus
- Address the same 5/6 aspects as we do in todays class!

Theory Task T3 (HS22)

An array of non-negative integers $A=[a_1,\dots,a_n]$ is called summy if and only if, for all $i\in\{2,\dots,n\}$, there exists a (possibly empty) set $I\subseteq\{1,\dots,i-1\}$ such that $a_i=\sum_{j\in I} a_j$

⇒ Every integer except the first one must be a ~~sum~~ the sum of (distinct) integers that precede it in the array.

Examples: • $[2,2,4,6,0,12]$ is summy, because $2=2$, $4=2+2$, $6=2+4$, $12=2+4+6$

• $[2,2,4,6,0,13]$ is not summy, since 13 can't be written as a sum of integers from $\{2,2,4,6,0\}$

Input: Array A of length n

Output: True if the array is summy, and False otherwise

For full points: $O(n\cdot maxA)$ runtime, where $maxA$ is the max. value of entries in A

Address the following aspects (given in exam):

1.) Definition of DP-Table (dimensions/meaning of entry, indexing!)

2.) Computation of an entry (base cases/computation of other entries)

3.) Calculation Order

4.) Extracting the solution

5.) Running time: Provide in Θ-notation in terms of n and $maxA$, justify your answer

Approach: • Understand the examples
• Look at dependencies ⇒ extract dimensions for DP
• Find DP-Recursion
• Rest

Size of DP-Table / Number of entries: DP-Table is 2-dimensional with indices $\{0,\dots,n\}\times\{0,\dots,maxA\}$. The total number of entries is $(n+1)\cdot(maxA+1)$

Meaning of DP-Table: $DP[i][j]$ is true if and only if there Exists $I\subseteq\{1,\dots i\}$ such that $j=\sum_{k\in I} a_k$

Computation of an entry (initialization and recursion):

• $DP[0][j] = (j==0)$

• $DP[i][0] = True$

out-of-bounds check

• $DP[i][j] = DP[i-1][j] \,||\, (j-a_i\geq 0 \;\&\&\; DP[i-1][j-a_i])$

Order of computation: In increasing oder of i and j e.g.

```
for i = 0, ..., n
    for j = 0, ..., maxA
        DP[i][j] = ...
```

Extracting the result: Check for all a_i with $i\geq 2$ if $DP[i-1][a_i]==True$ (logical and between all) ⇒ result is $\bigwedge_{i=2} DP[i-1][a_i]$

e.g.

```
for i=1, ..., n-1
    if (DP[i][A[i]] == false) return false
return true
```

Running time: • Initialization takes $\Theta(n)+\Theta(maxA)$

• Computation of an entry takes $O(1)$, we have $(n+1)\cdot(maxA+1)$ entries ⇒ $\Theta(n\cdot maxA)$

• Extracting the result takes $\Theta(n)$

⇒ Total runtime is $\Theta(n)+\Theta(maxA)+\Theta(n\cdot maxA)+\Theta(n) = \Theta(n\cdot maxA)$

Example with A=[2,2,4,6,0] (should return true)

Base Cases:

a_i	i \ j	0	1	2	3	4	5	6
(0)	0	T	F	F	F	F	F	F
2	1	T						
2	2	T						
4	3	T						
6	4	T						
0	5	T						

After 2nd row is computed →

a_i	i \ j	0	1	2	3	4	5	6
(0)	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
2	2	T						
4	3	T						
6	4	T						
0	5	T						

After 3rd row is computed →

a_i	i \ j	0	1	2	3	4	5	6
(0)	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
2	2	T	F	T	F	T	F	F
4	3	T						
6	4	T						
0	5	T						

After 4th row is computed →

a_i	i \ j	0	1	2	3	4	5	6
(0)	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
2	2	T	F	T	F	T	F	F
4	3	T	F	T	F	T	F	T
6	4	T						
0	5	T						

After 5th row is computed →

a_i	i \ j	0	1	2	3	4	5	6
(0)	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
2	2	T	F	T	F	T	F	F
4	3	T	F	T	F	T	F	T
6	4	T	F	T	F	T	F	T
0	5	T						

After 6th row is computed →

a_i	i \ j	0	1	2	3	4	5	6
(0)	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
2	2	T	F	T	F	T	F	F
4	3	T	F	T	F	T	F	T
6	4	T	F	T	F	T	F	T
0	5	T	F	T	F	T	F	T

Extracting the result:

a_i	i \ j	0	1	2	3	4	5	6
(0)	0	T	F	F	F	F	F	F
2	1	T	F	T a_2	F	F	F	F
2	2	T	F	T	F	T a_3	F	F
4	3	T	F	T	F	T	F	T a_4
6	4	T a_5	F	T	F	T	F	T
0	5	T	F	T	F	T	F	T

Theory Task T3 (FS21)

Given a square Matrix $M \in \mathbb{N}^{n \times n}$ of non-negative integers, with $n \geq 1$, the goal is to compute the longest snake within it.
The top-left element is M_{11}. M_{ij} denotes the j-th entry in the i-th row.

A snake is a sequence of entries M, i.e., $s = (s_1, \dots, s_k)$ (with $s_\ell = M_{i_\ell, j_\ell}$) with $i_\ell, j_\ell \in \{1, \dots, n\}$ and $\ell \in \{1, \dots, k\}$, that satisfies that the next entry s_{m+1} is either below the current one or to its left. This means that there are no cycles. Additionally, the corresponding matrix entries are required to differ exactly by 1, i.e., s satisfies $|s_m - s_{m+1}| = 1$, for all $m \in \{1, \dots, k-1\}$. The length of a snake $s = (s_1, \dots, s_k)$ is equal to its sequence length, which in this case is k.

Input: Matrix $M \in \mathbb{N}^{n \times n}$ of non negative integers
Output: Longest snake $s = (s_1, \dots, s_k)$. If there are multiple options return only one of them.

Example:

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

For full points: $O(n^2)$ runtime

Address the following aspects (given in exam): (same as prev. task)

Size of DP-Table / Number of entries: DP-Table is 2-dimensional with indices $\{1, \dots n\} \times \{1, \dots, n\}$. The total number of entries is n^2. 1-indexing!

Meaning of DP-Table: DP[i][j] contains the length of the longest snake that ends at M_{ij}

Computation of an entry (initialization and recursion): DP[1][n] = 1
all other entries (assuming we check for out of bounds):

$$DP[i][j] \begin{cases} DP[i][j+1]+1, & \text{if } |M_{i,j+1} - M_{i,j}| = 1 \text{ and } |M_{i-1,j} - M_{i,j}| \neq 1 \\ DP[i-1][j]+1 & \text{if } |M_{i,j+1} - M_{i,j}| \neq 1 \text{ and } |M_{i-1,j} - M_{i,j}| = 1 \\ \max(DP[i][j+1], DP[i-1][j])+1 & \text{if } |M_{i,j+1} - M_{i,j}| = 1 \text{ and } |M_{i-1,j} - M_{i,j}| = 1 \\ 1 & \text{else} \end{cases}$$

Order of computation: From top-right to bottom-left, so row-by-row, right to left, e.g for i=1, …, n
for j = n, … 1
DP[i][j] = …

Extracting the result: We extract the snake through backtracking.
- Look for largest entry in DP-Table, let's assume it is $k = DP[i][j]$. $\Rightarrow s_k = M_{i,j}$
- Backtrack $s_1, \dots, s_{k-1}$ by checking how we arrived at next sequenc entry.

$$\Rightarrow s_{k-1} = \begin{cases} M_{i-1,j} & \text{if } DP[i][j] = DP[i-1][j]+1 \text{ and } |M_{i,j} - M_{i-1,j}| = 1 \\ M_{i,j+1} & \text{if } DP[i][j] = DP[i][j+1]+1 \text{ and } |M_{i,j} - M_{i,j+1}| = 1 \end{cases}$$

and so on

Running time:
- Initialization takes $O(1)$
- Computation takes $O(1)$ per entry $\Rightarrow \Theta(n^2)$
- Backtracking takes $\Theta(n^2)$ (finding max) $+ O(1) \cdot \Theta(n)$ (backtracking) $\Rightarrow \Theta(n^2)$ total

$\Rightarrow$ Total runtime is $O(1) + \Theta(n^2) + \Theta(n^2) = \Theta(n^2)$

Example of DP-Table

M=

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

Base Case:

				1

After 1st row is computed ⟶

1	2	1	1	1

After 2nd row is computed ⟶

1	2	1	1	1
1	3	2	2	1

After 3rd row is computed ⟶

1	2	1	1	1
1	3	2	2	1
6	5	4	3	1

After 4th row is computed ⟶

1	2	1	1	1
1	3	2	2	1
6	5	4	3	1
7	6	5	4	1

After 5th row is computed ⟶

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

Backtracking

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