Informatik II

Übung 9

FS 2019

Program Today

1. Repetition theory
   - Editing Distance
2. In-Class Exercise
   - Implement on CodeExpert

1. Repetition theory

Dynamic Programming: Idea

- Divide a complex problem into a reasonable number of sub-problems
- The solution of the sub-problems will be used to solve the more complex problem
- Identical problems will be computed only once

Dynamic Programming Consequence

Identical problems will be computed only once

$\Rightarrow$ Results are saved

We trade spee against memory consumption

Dynamic Programming = Divide-And-Conquer ?

- In both cases the original problem can be solved (more easily) by utilizing the solutions of sub-problems. The problem provides *optimal substructure*.
- Divide-And-Conquer algorithms (such as Mergesort): sub-problems are independent; their solutions are required only once in the algorithm.
- DP: sub-problems are dependent. The problem is said to have *overlapping sub-problems* that are required multiple-times in the algorithm.
- In order to avoid redundant computations, results are tabulated. For *sub-problems there must not be any circular dependencies*.

Minimal Editing Distance

Editing distance of two sequences $A_n = (a_1, \ldots, a_n)$, $B_m = (b_1, \ldots, b_m)$.

Editing operations:

- Insertion of a character
- Deletion of a character
- Replacement of a character

Question: how many editing operations at least required in order to transform string A into string B.

TIGER ZIGER ZIEGER ZIEGE

Minimal Editing Distance

Wanted: cheapest character-wise transformation $A_n \rightarrow B_m$ with costs

operation	Levenshtein	LCS[1]	general
Insert c	1	1	ins(c)
Delete c	1	1	del(c)
Replace $c \rightarrow c'$	$\mathbb{1}(c \neq c')$	$\infty \cdot \mathbb{1}(c \neq c')$	repl(c, c')

Beispiel

T	I	G	E	R		T	I	_	G	E	R		T→Z	+E	-R
Z	I	E	G	E		Z	I	E	G	E	_		Z→T	-E	+R

[1] Longest common subsequence – A special case of an editing problem

Wie findet man den DP Algorithms

0. Exact formulation of the wanted solution
1. Define sub-problems (and compute the cardinality)
2. Guess / Enumerate (and determine the running time for guessing)
3. Recursion: relate sub-problems
4. Memoize / Tabularize. Determine the dependencies of the sub-problems
5. Solve the problem
   Running time = #sub-problems $\times$ time/sub-problem

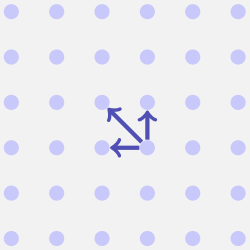
DP

0. $E(n, m)$ = mimimum number edit operations (ED cost) $a_{1 \ldots n} \to b_{1 \ldots m}$
1. Subproblems $E(i, j)$ = ED von $a_{1 \ldots i}$. $b_{1 \ldots j}$. #SP $= n \cdot m$
2. Guess Costs$\Theta(1)$
   - $a_{1..i} \to a_{1 \ldots i-1}$ (delete)
   - $a_{1..i} \to a_{1 \ldots i} b_j$ (insert)
   - $a_{1..i} \to a_{1 \ldots i_1} b_j$ (replace)
3. Rekursion

$$E(i, j) = \min \begin{cases} \text{del}(a_i) + E(i-1, j), \\ \text{ins}(b_j) + E(i, j-1), \\ \text{repl}(a_i, b_j) + E(i-1, j-1) \end{cases}$$

DP

4. Dependencies



$\Rightarrow$ Computation from left top to bottom right. Row- or column-wise.

5. Solution in $E(n, m)$

Example (Levenshtein Distance)

$$E[i,j] \leftarrow \min\left\{E[i-1,j]+1, E[i,j-1]+1, E[i-1,j-1]+\mathbb{1}(a_i \neq b_j)\right\}$$

	$\emptyset$	Z	I	E	G	E
$\emptyset$	0	1	2	3	4	5
T	1	1	2	3	4	5
I	2	2	1	2	3	4
G	3	3	2	2	2	3
E	4	4	3	2	3	2
R	5	5	4	3	3	3

Editing steps: from bottom right to top left, following the recursion.

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Computation of an entry**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Calculation order**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- **Extracting the solution**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- **Extracting the solution**: How can the final solution be extracted once the table has been filled?

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- **Extracting the solution**: How can the final solution be extracted once the table has been filled?

Bottom-Up DP algorithm ED

1 **Dimension of the table? Semantics?**

Bottom-Up DP algorithm ED

Dimension of the table? Semantics?

1. Table $E[0,\ldots,n][0,\ldots,m]$. $E[i,j]$: minimal edit distance of the strings $(a_1,\ldots,a_i)$ and $(b_1,\ldots,b_j)$

Bottom-Up DP algorithm ED

Dimension of the table? Semantics?

1 Table $E[0,\dots,n][0,\dots,m]$. $E[i,j]$: minimal edit distance of the strings $(a_1,\dots,a_i)$ and $(b_1,\dots,b_j)$

Computation of an entry

2

Bottom-Up DP algorithm ED

Dimension of the table? Semantics?

1. Table $E[0,\ldots,n][0,\ldots,m]$. $E[i,j]$: minimal edit distance of the strings $(a_1,\ldots,a_i)$ and $(b_1,\ldots,b_j)$

Computation of an entry

2. $E[0,i] \leftarrow i \; \forall 0 \le i \le m$, $E[j,0] \leftarrow i \; \forall 0 \le j \le n$.
otherwise via $E[i,j] =$
$\min\{\mathsf{del}(a_i) + E(i-1,j), \mathsf{ins}(b_j) + E(i,j-1), \mathsf{repl}(a_i,b_j) + E(i-1,j-1)\}$

Bottom-Up DP algorithm ED

3 **Computation order**

Bottom-Up DP algorithm ED

3 **Computation order**

Rows increasing and within columns increasing (or the other way round).

Bottom-Up DP algorithm ED

3 **Computation order**

Rows increasing and within columns increasing (or the other way round).

4 **Reconstruct solution?**

Bottom-Up DP algorithm ED

3 **Computation order**

Rows increasing and within columns increasing (or the other way round).

4 **Reconstruct solution?**

Start with $j = m$, $i = n$. If $E[i, j] = \text{repl}(a_i, b_j) + E(i-1, j-1)$ then output $a_i \to b_j$ and continue with $(j, i) \leftarrow (j-1, i-1)$; otherwise, if $E[i, j] = \text{del}(a_i) + E(i-1, j)$ output $\text{del}(a_i)$ and continue with $j \leftarrow j-1$ otherwise, if $E[i, j] = \text{ins}(b_j) + E(i, j-1)$, continue with $i \leftarrow i-1$. Terminate for $i = 0$ and $j = 0$.

Longest ascending Sequence in matrix

Given $n \times m$ matrix A:

9	27	42	41	48
35	39	8	3	5
12	49	2	38	4
15	47	29	28	6
19	1	25	33	10

Longest ascending Sequence in matrix

Given $n \times m$ matrix A:

9	27	42	41	48
35	39	8	3	5
12	49	2	38	4
15	47	29	28	6
19	1	25	33	10

Wanted longest ascending sequence:

$$4, 6, 28, 29, 47, 49$$

Definition of the DP table

- What are the dimensions of the table?

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m$

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m(\times 2)$

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m(\times 2)$
- What is the meaning of each entry?

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m(\times 2)$
- What is the meaning of each entry?
  - In $T[x][y]$ is the length of the longest ascending sequence that ends in $A[x][y]$
  - In $S[x][y]$ are the coordinates of the predecessor in ascending sequence (if exists)

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?
  - Consider neighbors with smaller entry in A

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?
  - Consider neighbors with smaller entry in A
  - From the smaller entries choose entry with the largest entry in T

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?
  - Consider neighbors with smaller entry in A
  - From the smaller entries choose entry with the largest entry in T
  - Update T and S. (S gets coordinate from selected neighbor, T gets value from selected neighbor increased by one)

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?
  - Consider neighbors with smaller entry in A
  - From the smaller entries choose entry with the largest entry in T
  - Update T and S. (S gets coordinate from selected neighbor, T gets value from selected neighbor increased by one)

Calculation order

- In which order can entries be computed so that values needed for each entry have been determined in previous steps?

Calculation order

- In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- Bottom-Up: Start with smallest element in A and so on. (Means that one has to sort A)

Calculation order

- In which order can entries be computed so that values needed for each entry have been determined in previous steps?

- Bottom-Up: Start with smallest element in A and so on. (Means that one has to sort A)

- Recursively: Arbitrary order, if entry is already computed skip it otherwise compute for smaller neighbor recursively.

Extracting the solution

- How can the final solution be extracted once the table has been filled?

Extracting the solution

- How can the final solution be extracted once the table has been filled?
  - Consider all entries to find one with a longest sequence. From there, we can reconstruct the solution by following the corresponding predecessors.

Program

Implement a DP solution in the prepared CodeExpert program.

Questions / Suggestions?