

17. Dynamic Programming II

Editing Distance, Bellman-Ford Algorithm
[Cormen et al, Kap. 24.1]]

Minimal Editing Distance

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

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 ²⁴	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

²⁴Longest common subsequence – A special case of an editing problem

DP

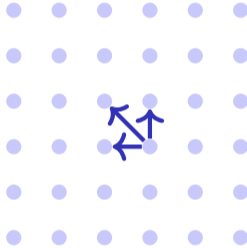
0. $E(n, m)$ = minimum number edit operations (ED cost) $a_{1\dots n} \rightarrow b_{1\dots m}$
1. Subproblems $E(i, j)$ = ED von $a_{1\dots i}$. $b_{1\dots j}$. #SP = $n \cdot m$
2. Guess Costs $\Theta(1)$
 - $a_{1\dots i} \rightarrow a_{1\dots i-1}$ (delete)
 - $a_{1\dots i} \rightarrow a_{1\dots i}b_j$ (insert)
 - $a_{1\dots i} \rightarrow a_{1\dots i-1}b_j$ (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



⇒ Computation from left top to bottom right. Row- or column-wise.

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

Example (Levenshtein Distance)

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

	\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	1	2
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.
Bottom-Up description of the algorithm: exercise

Bottom-Up DP algorithm ED

Dimension of the table? Semantics?

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

Computation of an entry

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

Bottom-Up DP algorithm ED

Computation order

3.

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

Reconstruction of a solution?

4.

Start with $j = m, i = n$. If $E[i, j] = \text{repl}(a_i, b_j) + E(i - 1, j - 1)$ then output $a_i \rightarrow b_j$ 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 $j = 0$.

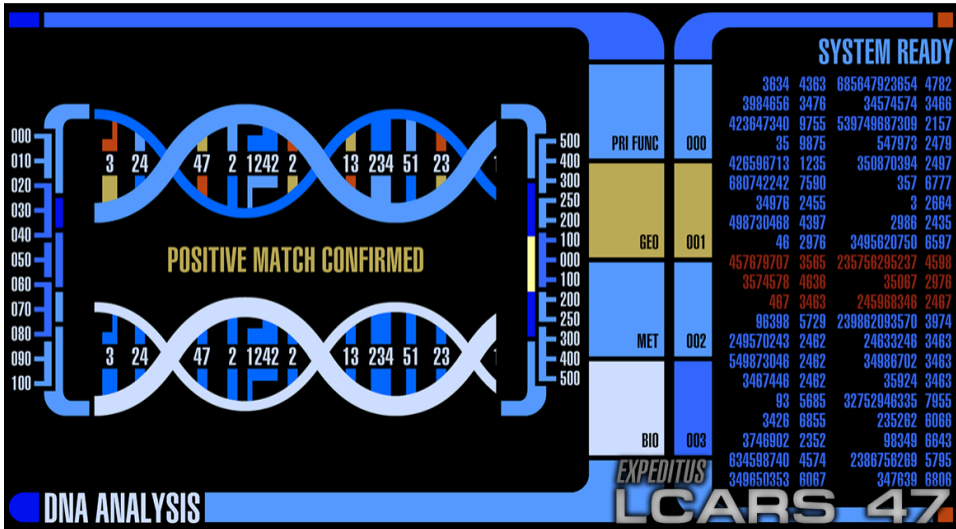
Analysis ED

- Number table entries: $(m + 1) \cdot (n + 1)$.
- Constant number of assignments and comparisons each. Number steps: $\mathcal{O}(mn)$
- Determination of solution: decrease i or j . Maximally $\mathcal{O}(n + m)$ steps.

Runtime overall:

$$\mathcal{O}(mn).$$

DNA - Comparison (Star Trek)



DNA - Comparison

- DNA consists of sequences of four different nucleotides **A**denine **G**uanine **T**hymine **C**ytosine
- DNA sequences (genes) thus can be described with strings of A, G, T and C.
- Possible comparison of two genes: determine the **longest common subsequence**

The longest common subsequence problem is a special case of the minimal edit distance problem.

Longest common subsequence

Subsequences of a string:

Subsequences(KUH): (), (*K*), (*U*), (*H*), (*KU*), (*KH*), (*UH*), (*KUH*)

Problem:

- **Input:** two strings $A = (a_1, \dots, a_m)$, $B = (b_1, \dots, b_n)$ with lengths $m > 0$ and $n > 0$.
- **Wanted:** Longest common subsequence (LCS) of A and B .

Longest Common Subsequence

Examples:

$LGT(IGEL, KATZE) = E$, $LGT(TIGER, ZIEGE) = IGE$

Ideas to solve?

T	I		G	E	R
Z	I	E	G	E	

Recursive Procedure

Assumption: solutions $L(i, j)$ known for $A[1, \dots, i]$ and $B[1, \dots, j]$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$, but not for $i = m$ and $j = n$.

T I E G E R
Z I E G E

Consider characters a_m, b_n . Three possibilities:

1. A is enlarged by one whitespace. $L(m, n) = L(m, n - 1)$
2. B is enlarged by one whitespace. $L(m, n) = L(m - 1, n)$
3. $L(m, n) = L(m - 1, n - 1) + \delta_{mn}$ with $\delta_{mn} = 1$ if $a_m = b_n$ and $\delta_{mn} = 0$ otherwise

Recursion

$$L(m, n) \leftarrow \max\{L(m-1, n-1) + \delta_{mn}, L(m, n-1), L(m-1, n)\}$$

for $m, n > 0$ and base cases $L(\cdot, 0) = 0, L(0, \cdot) = 0$.

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

Dynamic Programming algorithm LCS

Dimension of the table? Semantics?

1. Table $L[0, \dots, m][0, \dots, n]$. $L[i, j]$: length of a LCS of the strings (a_1, \dots, a_i) and (b_1, \dots, b_j)

Computation of an entry

2. $L[0, i] \leftarrow 0 \forall 0 \leq i \leq m, L[j, 0] \leftarrow 0 \forall 0 \leq j \leq n$. Computation of $L[i, j]$ otherwise via $L[i, j] = \max(L[i-1, j-1] + \delta_{ij}, L[i, j-1], L[i-1, j])$.

Dynamic Programming algorithm LCS

Computation order

3.

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

Reconstruction of a solution?

4.

Start with $j = m, i = n$. If $a_i = b_j$ then output a_i and continue with $(j, i) \leftarrow (j-1, i-1)$; otherwise, if $L[i, j] = L[i, j-1]$ continue with $j \leftarrow j-1$ otherwise, if $L[i, j] = L[i-1, j]$ continue with $i \leftarrow i-1$. Terminate for $i = 0$ or $j = 0$.

Analysis LCS

- Number table entries: $(m + 1) \cdot (n + 1)$.
- Constant number of assignments and comparisons each. Number steps: $\mathcal{O}(mn)$
- Determination of solution: decrease i or j . Maximally $\mathcal{O}(n + m)$ steps.

Runtime overall:

$$\mathcal{O}(mn).$$

Reminder: Shortest Path Algorithm

1. Initialise d_s and π_s : $d_s[v] = \infty$, $\pi_s[v] = \text{null}$ for each $v \in V$
2. Set $d_s[s] \leftarrow 0$
3. Choose an edge $(u, v) \in E$

Relaxiere (u, v) :

if $d_s[v] > d_s[u] + c(u, v)$ then

$d_s[v] \leftarrow d_s[u] + c(u, v)$

$\pi_s[v] \leftarrow u$

4. Repeat 3 until nothing can be relaxed any more.
(until $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$)

Dynamic Programming Approach (Bellman)

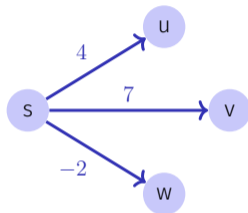
Induction over number of edges $d_s[i, v]$: Shortest path from s to v via maximally i edges.

$$d_s[i, v] = \min\{d_s[i - 1, v], \min_{(u,v) \in E} (d_s[i - 1, u] + c(u, v))\}$$

$$d_s[0, s] = 0, d_s[0, v] = \infty \quad \forall v \neq s.$$

Dynamic Programming Approach (Bellman)

	s	\dots	v	\dots	w
0	0	∞	∞	∞	∞
1	0	∞	7	∞	-2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
$n - 1$	0	\dots	\dots	\dots	\dots



Algorithm: Iterate over last row until the relaxation steps do not provide any further changes, maximally $n - 1$ iterations. If still changes, then there is no shortest path.

Algorithm Bellman-Ford(G, s)

Input: Graph $G = (V, E, c)$, starting point $s \in V$

Output: If return value true, minimal weights d for all shortest paths from s ,
otherwise no shortest path.

```
foreach  $u \in V$  do  
   $d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$   
 $d_s[s] \leftarrow 0;$   
for  $i \leftarrow 1$  to  $|V|$  do  
   $f \leftarrow \text{false}$   
  foreach  $(u, v) \in E$  do  
     $f \leftarrow f \vee \text{Relax}(u, v)$   
  if  $f = \text{false}$  then return true  
return false;
```