From Orthogonal Vectors to Longest Common Subsequence

Definition: Longest Common Subsequence Problem (LCS)
Input: Strings x, y of length $|x| = n \geq |y| = m$
Task: Compute longest string z that is a subsequence of both x and y
(We write $LCS(x, y) = |z|$)

Example:

A G C A T A G C A T
G A C G A C

Dynamic Programming Algorithm:

Idea: Build a table

	x[1]	...	x[n]
y[1]			
:			
y[n]			

$T[i,j] = LCS[x[1..i], y[1..j]]$
$(\Rightarrow T[n,m] = LCS(x,y))$

Determine new entry by computation on "surrounding" entries

$T[0,j] = 0$ and $T[i,0] = 0$

How to determine $T[i,j]$ for increasing $1 \leq i \leq n$ and $j \leq i \leq m$

~~If $x[i] = y[j]$:~~ "delete in x" "delete" in y

$T[i,j] = \max\{T[i-1,j], T[i,j-1]\}$

If $x[i] = x[j]$:

$T[i,j] = \max\{T[i,j], T[i-1,j-1] + 1\}$

match

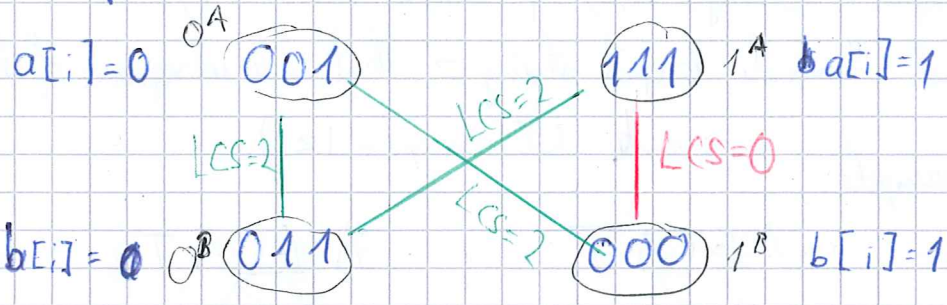
Time Complexity: $O(n^2)$
Fastest Known Algorithm: $O(n^2/\log^2 n)$ [Masek, Paterson '80]

Theorem: Assuming OVH, LCS has no algorithm with running time $O(n^{2-\delta})$ (for constant $\delta > 0$).
[Bringmann/Künnemann '15, Abboud/Backurs/V. Williams '15]

Given OV-instance $A, B \subseteq \{0,1\}^d$ $|A| = |B| = n$

Remember: $a \perp b \iff \forall i \; \forall 1 \le i \le d \; a[i] \cdot b[i] = 0$

Coordinate Gadgets: Simulate behavior of $a[i] \cdot b[i]$:



number $a[i] \longrightarrow$ string $a[i]^A$

$b[i] \longrightarrow$ string $b[i]^B$

Now $LCS(a[i]^A, b[i]^B)$ can be written as $f(a_i, b_i)$ with $f(0) \neq f(1)$

Vector Gadgets: Simulate orthogonality

Concatenate coordinate gadgets, pad with new symbol 2

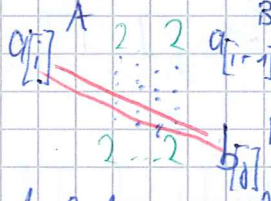
$VG(a) = a[1]^A \; 22..2 \; a[2]^A \; 22..2 \; a[3]^A \; 2..2 \; \ldots \; a[d]^A$

$VG(b) = b[1]^B \; 22..2 \; b[2]^B \; 2..2 \; b[3]^B \; 2..2 \; \ldots \; b[d]^B$

(length $4d$)

Claim: no LCS matches symbols in $a[i]^A$ with symbols in $b[j]^B$ when $i \neq j$

Proof: Assume otherwise (w.l.o.g. $i < j$)



We miss at least one possibility for matching two 2-sequences

Thus we will match at most $(d-2)$ $4d$ symbols 2 and $\le 3d$ symbols 0/1

But: $LCS(VG(a), VG(b)) \ge (d-1)4d > (d-2)4d + 3d$ ↯

&

Claim: Some LCS matches all 2's

Proof Sketch: Either 2 is skipped in both $VG(a)$ and $VG(b)$

2 1 2 ... 2 Then, can increase subsequence by matching

↓ × ↓

2 2 ...

Or 2 is skipped in only one of $VG(a)$ and $VG(b)$

2 2

↓ ↘

2 2 ... Then we can also increase length of subseq by matching

Conclusion: $LCS(VG(a), VG(b)) = (d-1)4d + \sum_{i=1}^{d} LCS(a_i^A, b_i^B)$

$= f(a_1, b_1)$

In particular, if $a_i \perp b_i$ for all i, then

$$\sum_{i=1}^{d} LCS(a_i^A, b_i^B) = d \cdot 2$$

Thus, $LCS(VG(a), VG(b)) = C+2$ if $a \perp b$

$\leq C$ o.w. Want: $= C$

for $C = (d-1)4d + 2d - 2$

Normalized Vector Gadgets:

Can always add a $(d+1)$st coordinate:

$a_{d+1} := 0$
$b_{d+1} := 1$ } does not influence if $a \perp b$

With this change we still have:

$\exists C$: $LCS(VG(a), VG(b)) = C+2$ if $a \perp b$

$\leq C$ o.w.

$(C = 4d^2 + 2d)$

Define vector $s := (0, 0, \ldots, 0, 1) \in \{0,1\}^{d+1}$

Clearly: $LCS(VG(s), VG(b)) = C$

[all components orthogonal, except for comp. $d+1$]

Goal: actually solve $\max\{LCS(VG(a), VG(b)), LCS(VG(s), VG(b))\}$

$\parallel$ $\quad$ $\parallel$

$C+2$ if $a \perp b$ $\quad$ C o.w.

New vector gadgets $VG'(a)$ and $VG'(b)$

$VG'(a)$: $VG(a)\ 44\ldots4\ VG(s)$

$VG'(b)$: $44\ldots4\ VG(b)\ 44\ldots4$

fresh symbol, length $10d^2$

Observation:

Will either match $VG(a)\ 444\ldots$ with $VG(b)\ 4\ldots4$

or $44\ldots4\ VG(s)$ with $4\ldots4\ VG(b)$ (similar arguments as before)

Claim: $LCS(VG'(a), VG'(b)) = 10d^2 + \max\{LCS(VG(a), VG(b)), LCS(VG(a), VG(s))\}$

Thus: $\exists C'$ s.t. $LCS(VG'(a), VG'(b)) = \begin{cases} C'+2 & \text{if } a \perp b \\ C' & \text{o.w.} \end{cases}$ $(C' = C + 10d^2)$

OR-Gadget: Does there $a \in A, b \in B$ s.t. $a \perp b$?

Write $A = \{a_1, \dots, a_n\}$ $B = \{b_1, \dots, b_n\}$

Want to construct two strings as follows:

every vector gadget appears twice

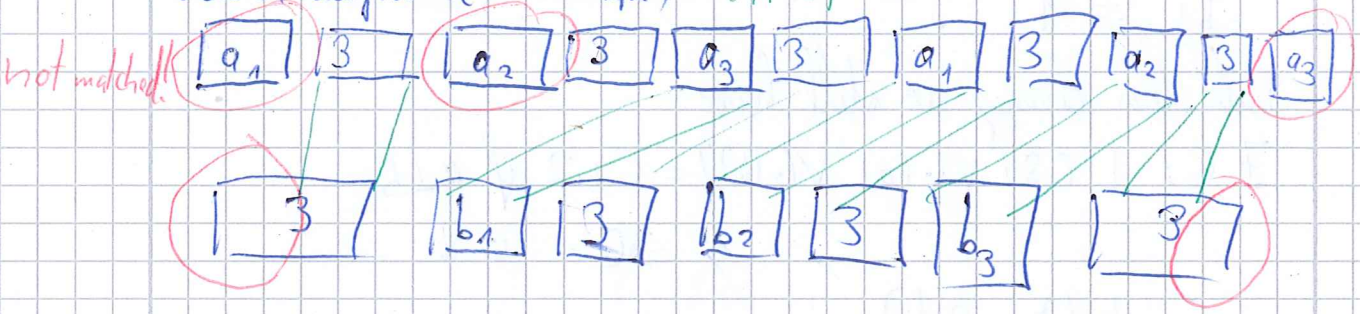
X: $VG'(a_1)$ 3..3 $VG'(a_2)$ 3..3 ... 3..3 $VG'(a_n)$ 3..3 $VG'(a_1)$ 3..3 ... $VG'(a_n)$

fresh symbol, length $100d^2$, length $100d^2 \cdot 2n$

Y: 33..3 $VG'(b_1)$ 3..3 $VG'(b_2)$ 3..3 ... 3..3 $VG'(b_n)$ 3..3

length $100d^2 \cdot 2n$, every vector gadget appears once (additional padding at front and end)

Desired alignment: (example) offset $\Delta = 2$



In general, if we align $VG'(b_j)$ with $VG'(a_{(\Delta+j) \bmod n})$ for some offset Δ

$LCS(x,y) \geq$ #3's in upper string (will all be matched) $+ \sum_{j=1}^{n} LCS(VG'(a_{(\Delta+j) \bmod n}), VG'(b_j))$

$= (2n-1) \cdot 100d^2$ $\geq (n-1) \cdot C' + (C'+2) = nC'+2$

$= (2n-1)\,100d^2 + nC' + 2$

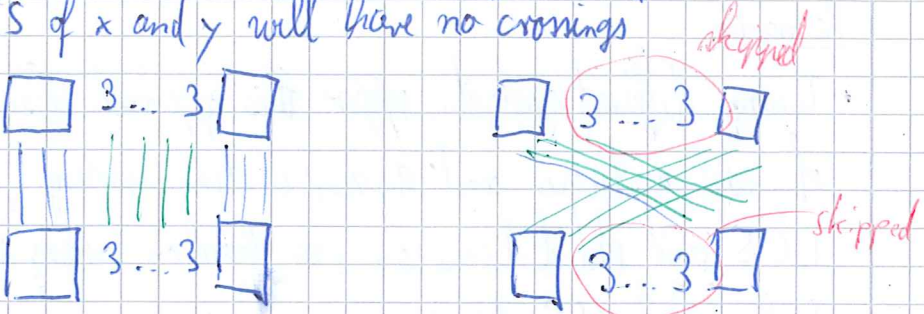
If there is an orthogonal pair, namely $a^* \in A, b^* \in B$, there is some offset Δ s.t. $VG'(a^*)$ and $VG'(b^*)$ will be aligned

(and then $LCS(VG'(a^*), VG'(b^*)) = C'+2$)

Remains to show:

If there is no orthogonal pair, then $LCS(x,y) \leq (2n-1)100d^2 + nC$

Observation: LCS of x and y will have no "crossings"

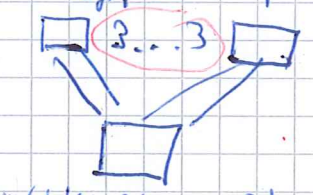


subsequence length $\geq 100d^2$ length $\leq 2\cdot(C'+2) < 100d^2$

We can now bound LCS in case of no orthogonal pair as follows:

$$LCS(x,y) \leq (2n-1)100d^2 + \sum_{j=1}^{n} \begin{cases} 0 & \text{if } VG'(b_j) \text{ is not matched} \\ C & \text{if } VG'(b_j) \text{ is matched to one } VG'(a_i) \\ |VG'(b_j)| - |3\ldots3| & \text{if } VG'(b_j) \text{ is matched to several } VG'(a_i)\text{'s} \end{cases}$$

($|VG'(b_j)| - |3\ldots3| \leq 0$)



$|VG'(b_j)| - |3\ldots3|$

$\leq 100d^2$ $\geq 100d^2$

$$\leq (2n-1)100d^2 + n\cdot C$$

We have thus proved the following:

Lemma: A, B contains orthogonal pair iff $LCS(x,y) = (2n-1)100d^2 + nC' + 2$

If A, B contains ~~orth~~

More precisely: If A, B contains orthogonal pair, then $LCS(x,y) = \ldots + 2$

If A, B does not contain OP, then $LCS(x,y) = \ldots$

Time Complexity: $|y| \leq |x| \leq O(d^2 n)$

(Time to construct $|x|, |y|$: $O(d^2 n)$)

Thus, if there were an $O(n^{2-\delta})$-algorithm for LCS (as $n = \max\{|x|, |y|\}$), then there would be an

$O((d^2 n)^{2-\delta}) = O(n^{2-\delta}\,\text{poly}(d))$ ($n = |A| = |B|$) algorithm for the OV Problem

Theorem: assuming OVH, there is no $O(n^{2-\delta})$ algorithm (for constant $\delta > 0$) for the LCS problem.

Remarks:

- Using different gadgets, this approach also shows hardness of edit distance and dynamic time warping
- LCS and edit distance are even hard on binary strings (alphabet: $\{0, 1\}$)