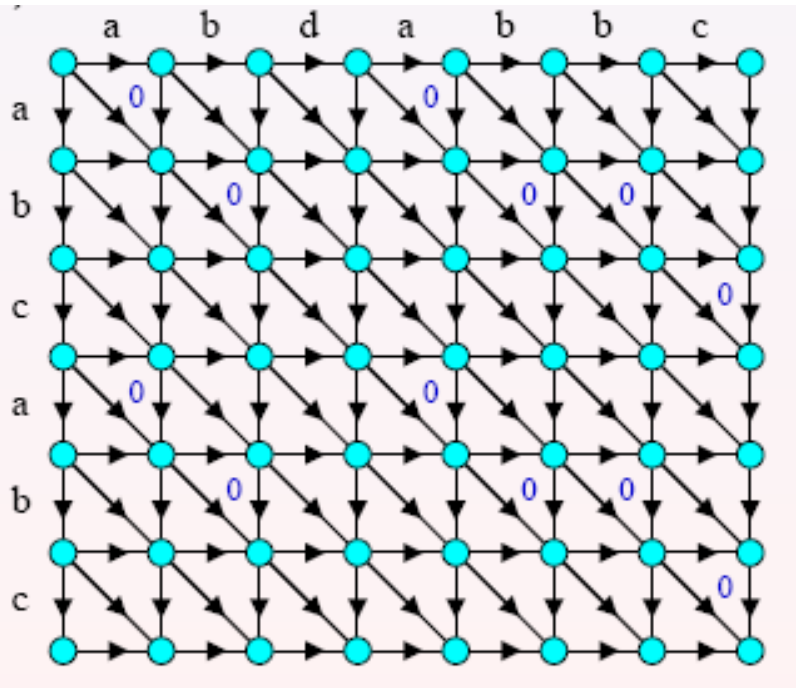


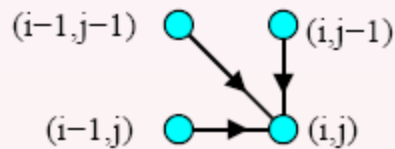
Alignment Graph



Alignment Matrix

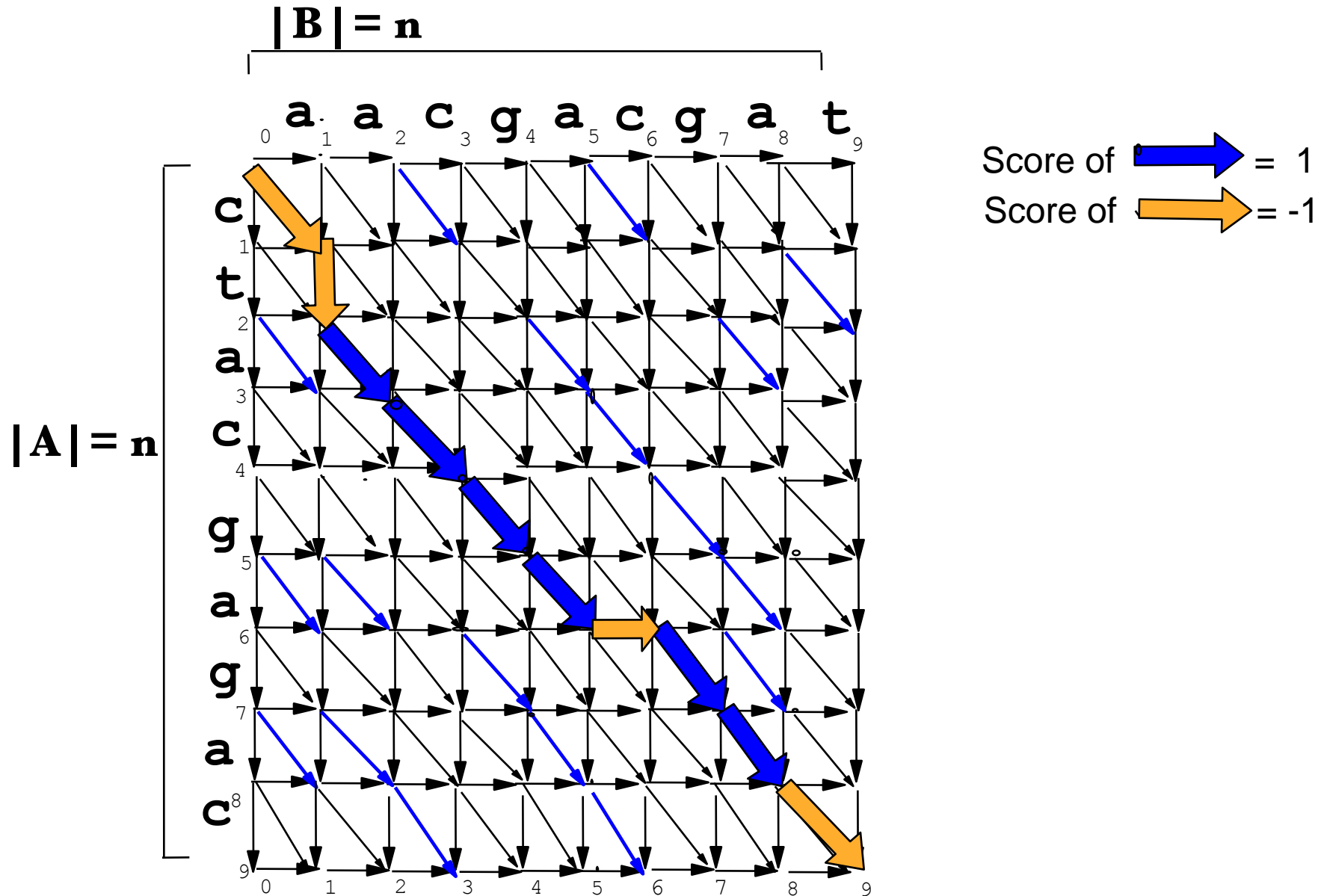
	a	b	d	a	b	b	c
a	0	1	2	3	4	5	6
b	1	0	1	2	3	4	5
c	2	1	0	1	2	3	4
a	3	2	1	1	2	3	4
b	4	3	2	2	1	2	3
c	5	4	3	3	2	1	2
c	6	5	4	4	3	2	2

• For $i, j > 0$:



$$F(i, j) = \min \left\{ \begin{array}{l} F(i-1, j) + 1, \\ F(i, j-1) + 1, \\ F(i-1, j-1) + w(i, j) \end{array} \right\}$$

Computing the Optimal Global Alignment Value



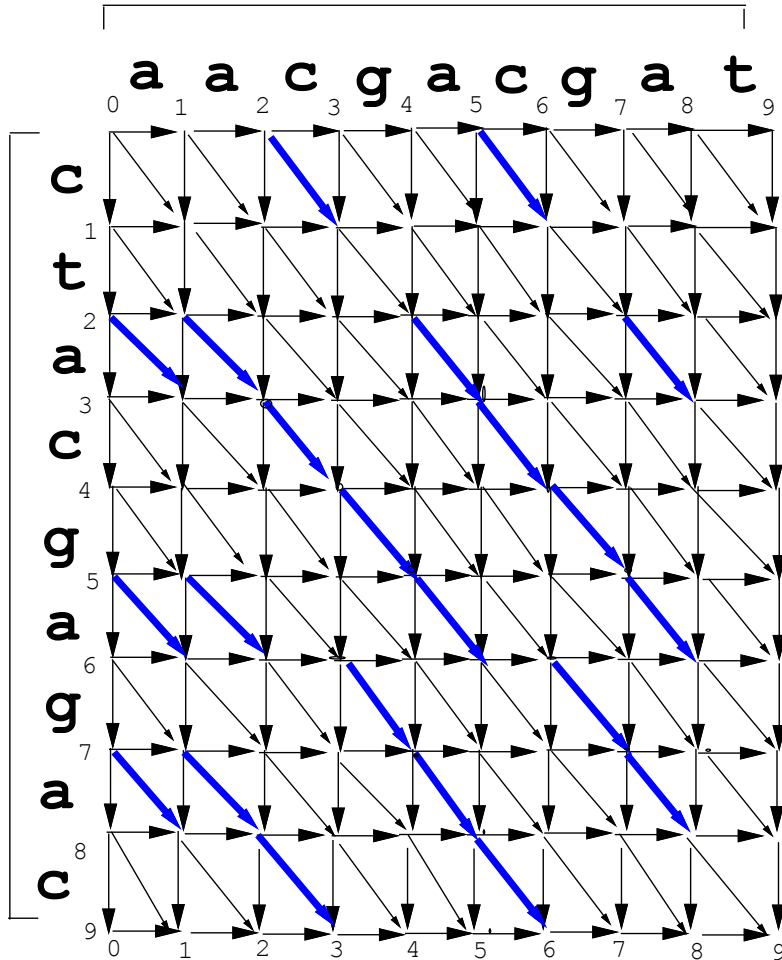
Classical Dynamic Programming: $O(n^2)$

The $O(n^2)$ time, Classical Dynamic Programming Algorithm

The Alignment Graph

$|B| = n$

$|A| = n$

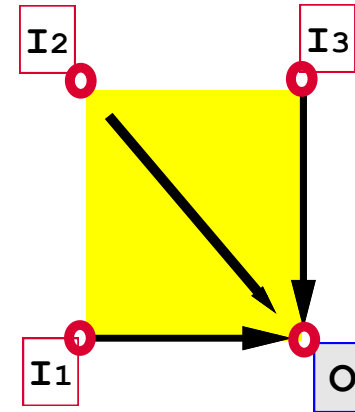


A: a c g

B: a c g

A: a c g a

B: a c g



A: a c g

B: a c g a

A: a c g a

B: a c g a

$$O = \max_{x=1}^3 (I_x + \text{edge}[I_x, O])$$

Can the quadratic complexity of the optimal alignment value computation be reduced **without relaxing the problem?**

Is it Possible to Align Sequences in Subquadratic Time?

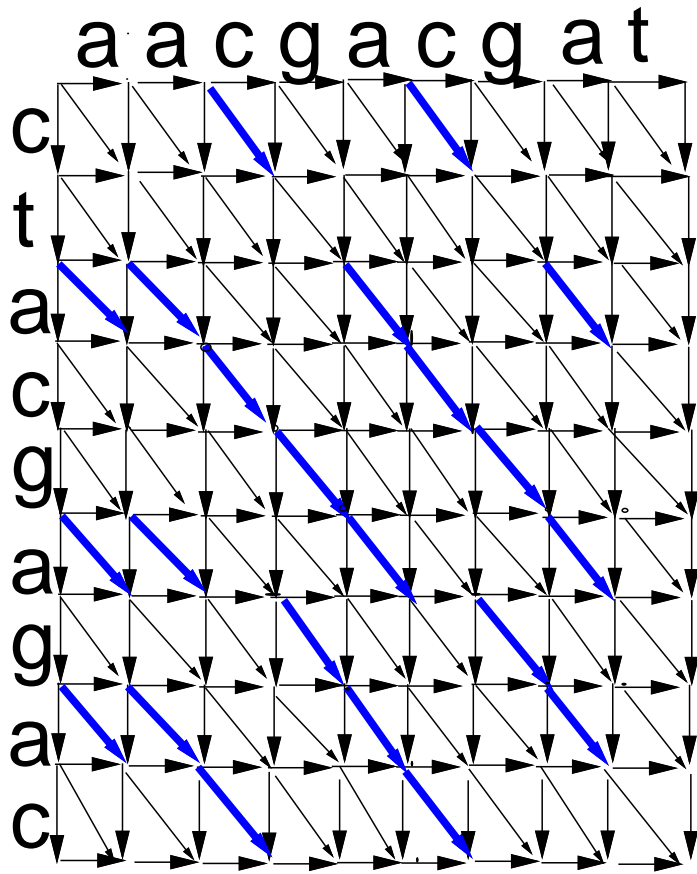
- Dynamic Programming takes $O(n^2)$ for global alignment
- Can we do better? $O(h n^2 / \log n)$, $h \leq 1$.

Landau, Crochemore & Ziv-ukelson 2003

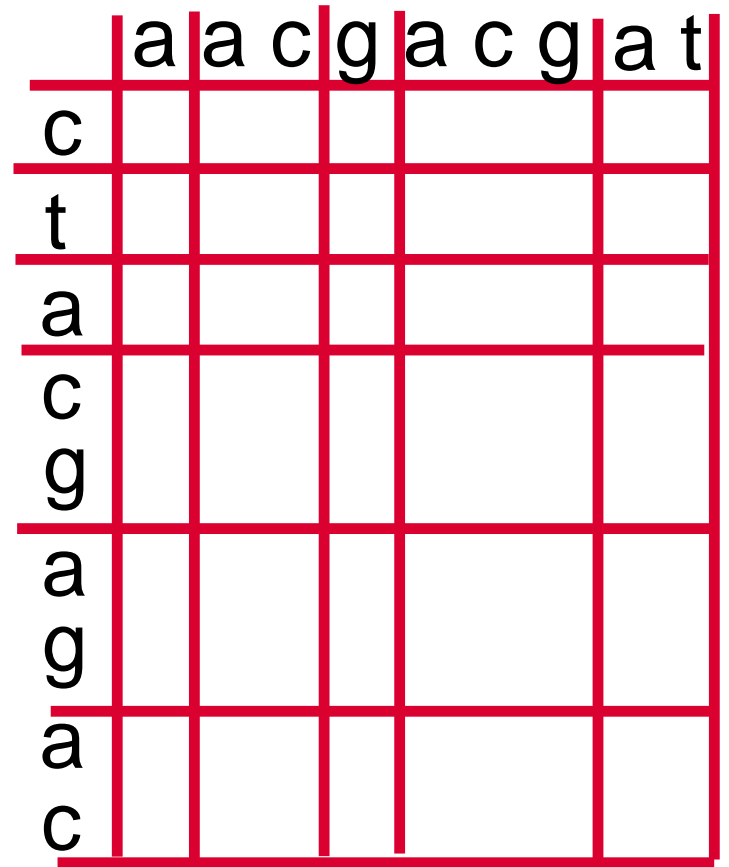
Techniques:

(1) **Compress** the sequences.

(2) Utilize the **Total Monotonicity** of DIST.

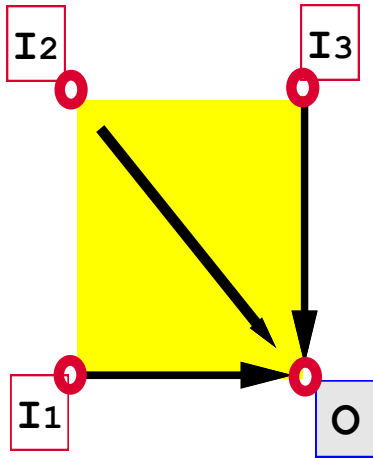


$O(n^2)$ vertices



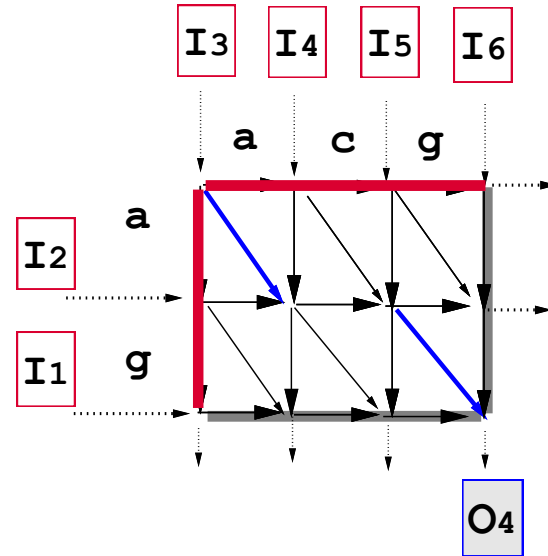
$O(h n / \log n)$ rows of n vertices +
 $O(h n / \log n)$ columns of n vertices

Standard, single-cell DP



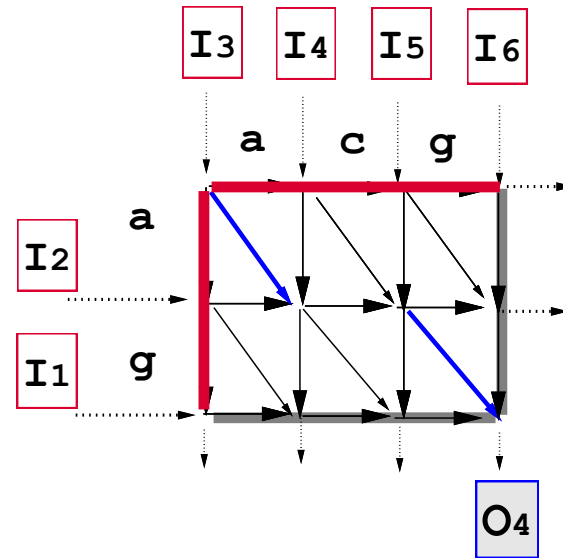
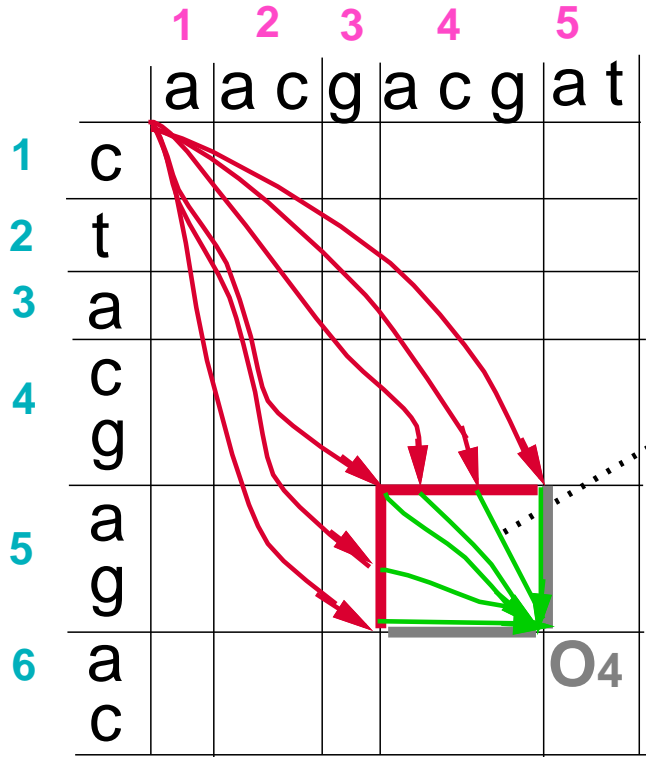
$$O = \max_{x=1}^3 (I_x + \text{edge}[I_x, O])$$

New, extended-cell DP



$$O_4 = \max_{x=0}^6 (I_x + \text{DIST}[x, 3])$$

Computing the score for Output Border Vertex O_4



I_1	$DIST[1, 4]$	$I_1 + DIST[1, 4]$
I_2	$DIST[2, 4]$	$I_2 + DIST[2, 4]$
I_3	$DIST[3, 4]$	$I_3 + DIST[3, 4]$
I_4	$DIST[4, 4]$	$I_4 + DIST[4, 4]$
I_5	$DIST[5, 4]$	$I_5 + DIST[5, 4]$
I_6	$DIST[6, 4]$	$I_6 + DIST[6, 4]$

$$O_4 = \max_{x=0}^6 (I_x + DIST[x, 3])$$

1 2 3 4 5 6

Input I \ DIST Matrix

I1 = 1	0	-1	-2	-3	□	□
I2 = 2	-1	-1	-2	-1	-3	□
I3 = 3	-2	0	0	1	-1	-3
I4 = 2	□	-2	-2	0	-2	-2
I5 = 1	□	□	-2	0	-1	-1
I6 = 3	□	□	□	-2	-1	0

OUT[x, j] = Ix + DIST[x, j]

1	0	-1	-2	□	□
1	1	0	1	-1	□
①	③	③	④	2	0
□	0	0	2	0	0
□	□	-1	1	0	0
□	□	□	1	②	③

Output vector O

1 3 3 4 2 3

The Main Challenges

How to obtain the DIST for G in O(t) time ?

(Take advantage of the incremental nature of LZ78 parsing).

How to compute the column maxima of OUT in O(t) time ?

(Utilize the Total Monotonicity Property of OUT).

How does Total Monotonicity affect Column Maxima behavior?

For all $a < b$ and $c < d$, $OUT[a,c] \leq OUT[b,c] \Rightarrow OUT[a,d] \leq OUT[b,d]$

OUT Matrix

				c	d		
a	1	0	-1	-2	$-\omega$	$-\omega$	
	1	1	0	1	-1	$-\omega$	
b	1	3	3	4	2	0	
	-12	0	0	2	0	0	
	-13	-13	-1	1	0	0	
	-14	-14	-14	1	2	3	

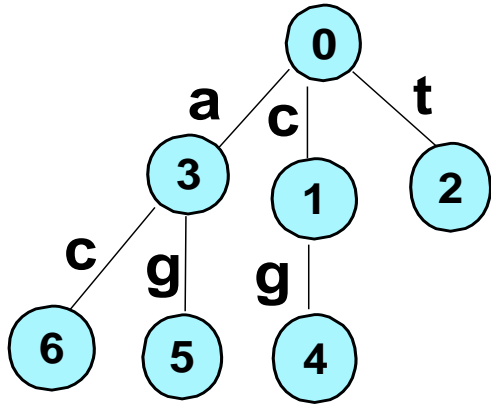
Column maxima row indices are monotonically non-decreasing.

SMAWK Matrix Searching[Aggarwal et-al 87] .

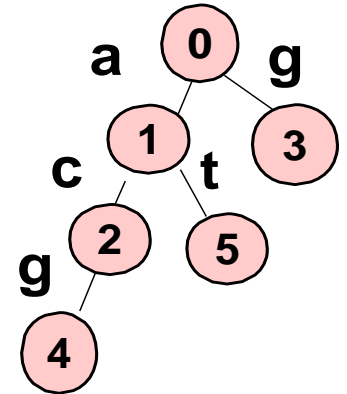
The **t** column maxima of a Totally Monotone array can be computed in $O(t)$ time, by querying only $O(t)$ elements.

Accessing a Prefix Block in Constant time.

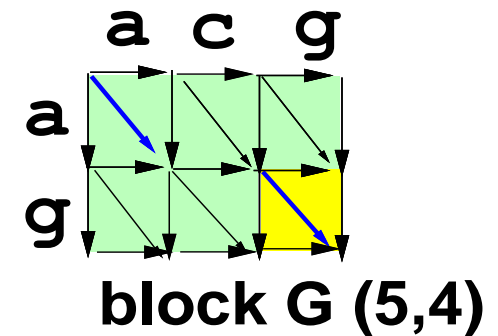
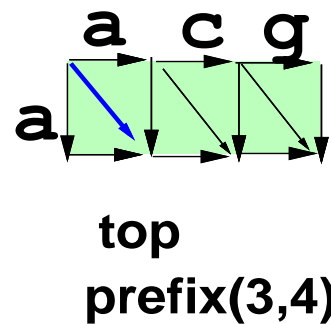
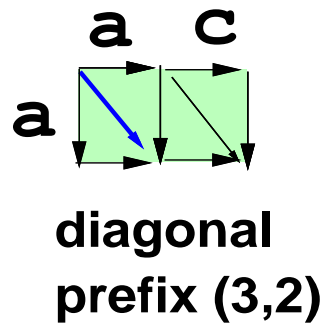
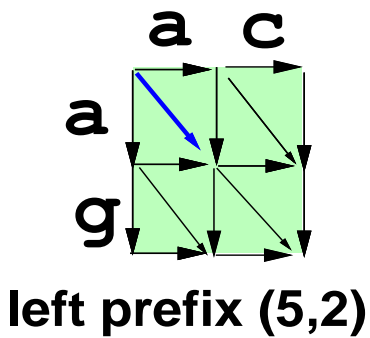
Trie for A



Trie for B



	1	2	3	4	5				
1	a	a	c	g	a	c	g	a	t
2	c								
3	t								
4	a	3/2		3/4					
5	c								
6	g								
7	a	5/2		5/4					
8	g								
9	a								
10	c								



Another technique to Align Sequences in Subquadratic Time?

- For **limited edit scoring schemes**, such as LCS, use “*Four-Russians*” Speedup

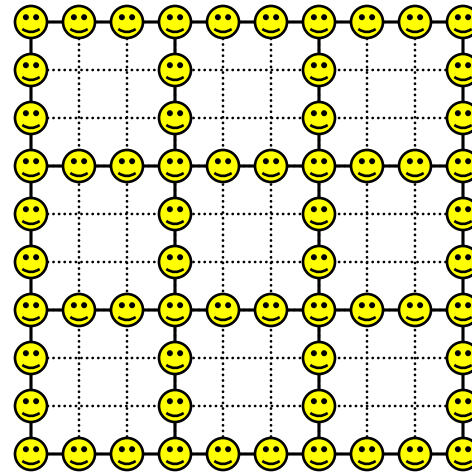
How Many Points Of Interest?

LZ-78 compression



$O(h n / \log n)$ rows of n vertices +
 $O(h n / \log n)$ columns of n vertices

blocks of size t



How many points of interest? $O(n^2/t)$

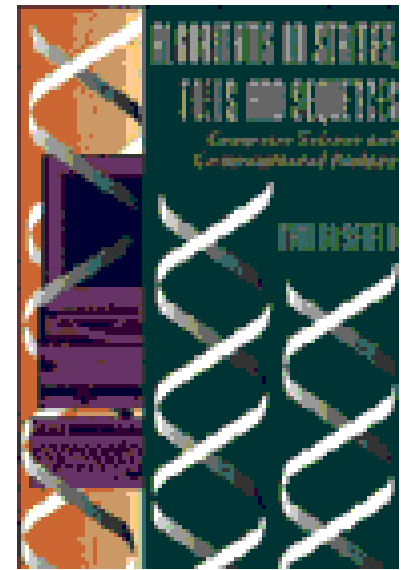
n/t rows with n vertices each

n/t columns with n vertices each

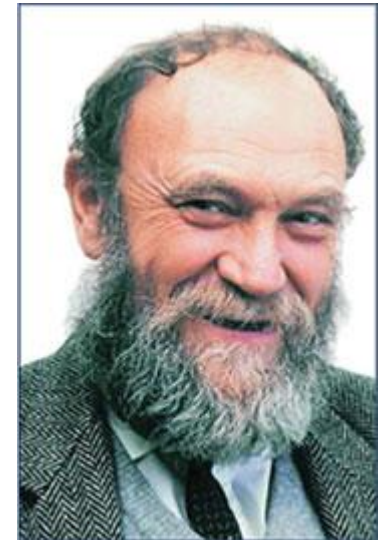
The “*Four-Russians*” technique for speeding up for dynamic programming

Dan Gusfield: The idea, comes from a paper by four authors ... concerning boolean matrix multiplication.

The general idea taken from this paper has come to be known in the West as The Four-Russians technique, even though only one of the authors is Russian.

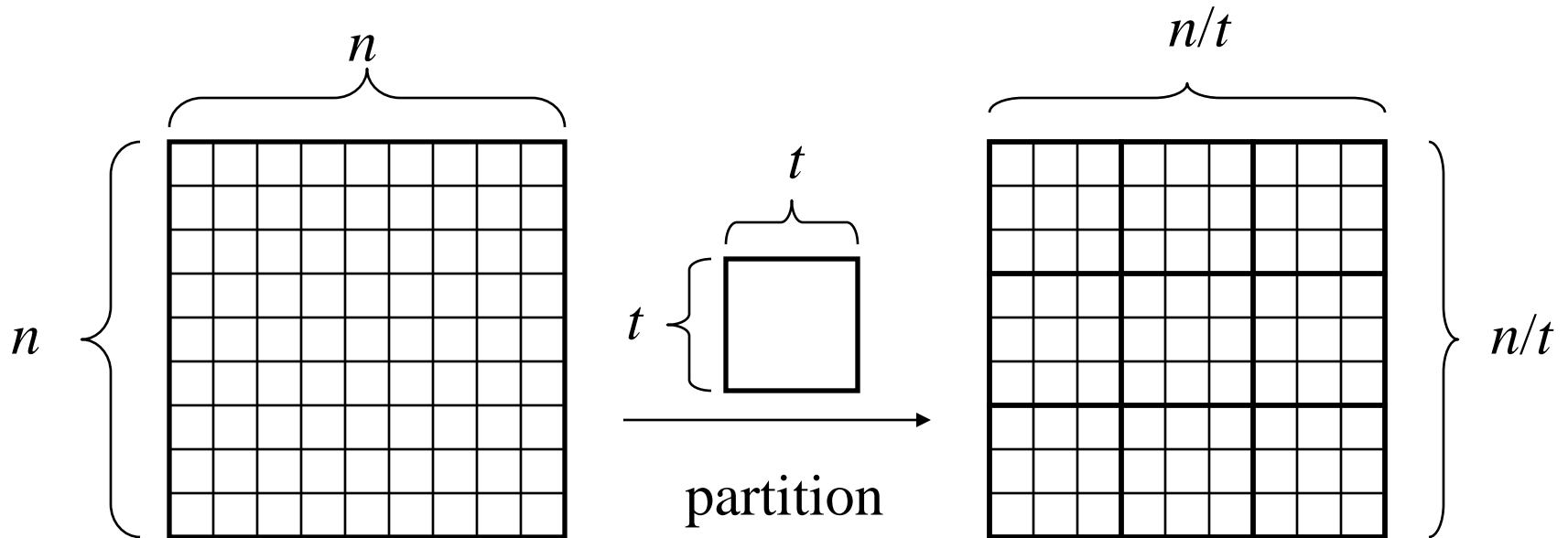


Arlazarov, Dinic, Kronrod and Faradzev



Masek & Paterson applied the “Four Russians” to the string edit problem

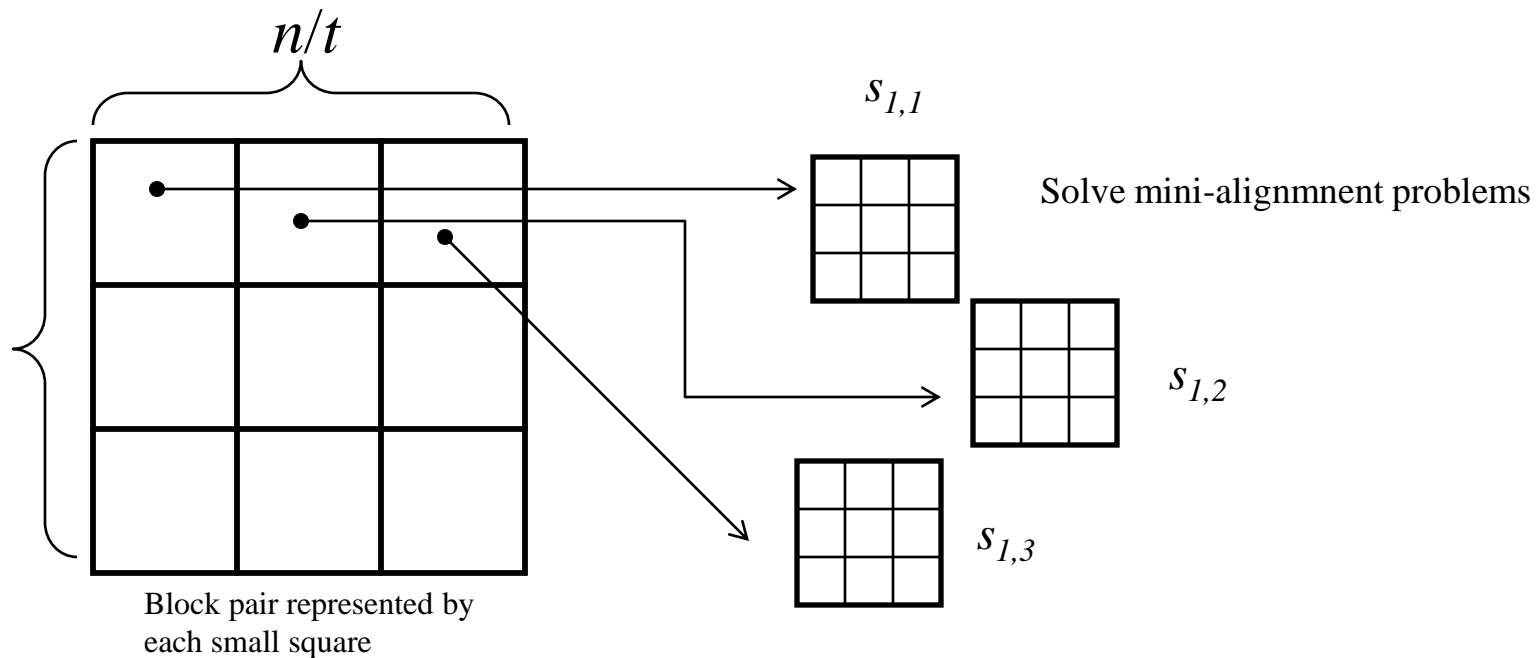
Partitioning Alignment Grid into Blocks of equal size t



Block Alignment Problem

- Goal: Find the longest block path through an edit graph
 - Input: Two sequences, u and v partitioned into blocks of size t . This is equivalent to an $n \times n$ edit graph partitioned into $t \times t$ subgrids
 - Output: The block alignment of u and v with the maximum score (longest block path through the edit graph)
-

Stage 1: compute the mini-alignments



How many blocks?
 $(n/t) * (n/t) = (n^2/t^2)$

Stage 2: dynamic programming

- Let $s_{i,j}$ denote the optimal block alignment score between the first i blocks of \mathbf{u} and first j blocks of \mathbf{v}

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j} - \sigma_{\text{block}} \\ s_{i,j-1} - \sigma_{\text{block}} \\ s_{i-1,j-1} + \beta_{i,j} \end{array} \right.$$

σ_{block} is the penalty for inserting or deleting an entire block

$\beta_{i,j}$ is score of pair of blocks in row i and column j .

Block Alignment Runtime

- Indices i, j range from 0 to n/t

- Running time of algorithm is

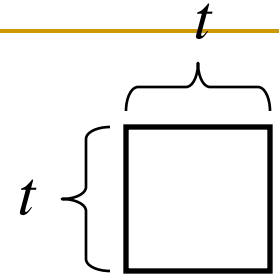
$$O([n/t] * [n/t]) = O(n^2/t^2)$$

if we don't count the time to compute each $\beta_{i,j}$

Block Alignment Runtime (cont'd)

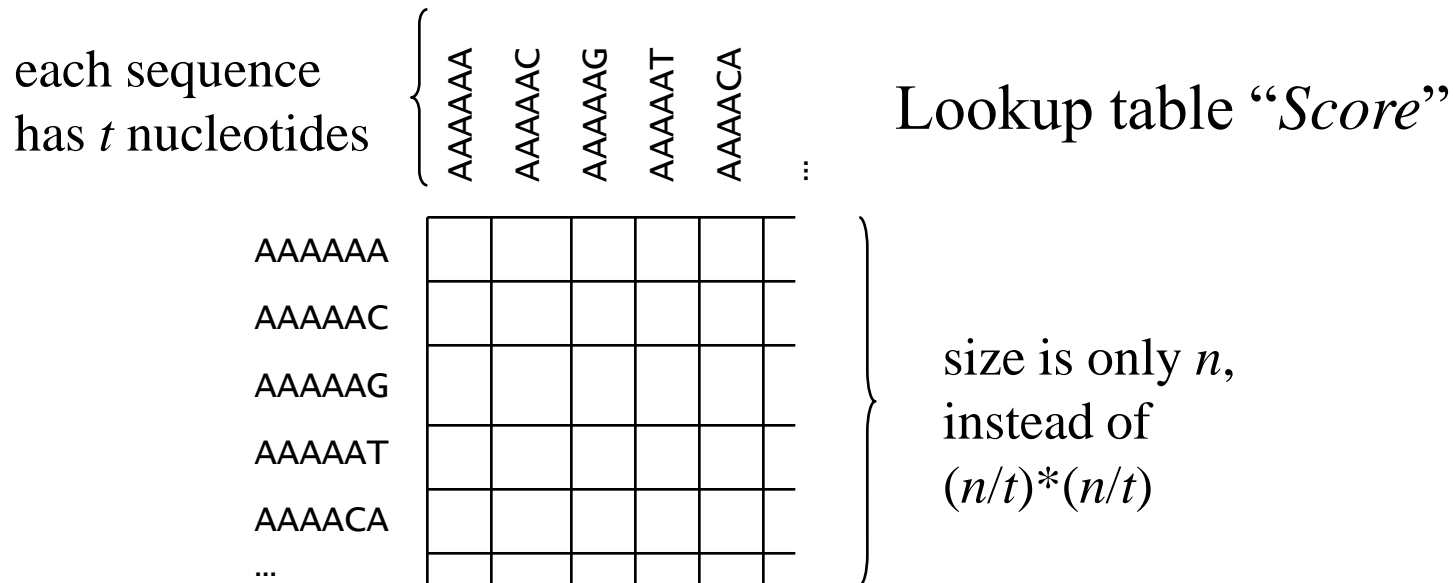
- Computing all $\beta_{i,j}$ requires solving $(n/t) * (n/t) = n^2/t^2$ mini block alignments, each of size $(t * t) = t^2$
- So computing all $\beta_{i,j}$ takes time $O(n^2/t^2 * t^2) = O(n^2)$
- This is the same as dynamic programming
- How do we speed this up?

Four Russians Technique



- Let $t = \log(n)$, where t is block size, n is sequence size.
- Instead of having $(n/t)^*(n/t) = n^2/t^2$ mini-alignments, construct $4^t \times 4^t$ mini-alignments for all pairs of strings of t nucleotides (huge size), and put in a lookup table.
- However, size of lookup table is not really that huge if t is small. Let $t = (\log n)/4$. Then $4^t \times 4^t = 4^{(\log n)/4} \times 4^{(\log n)/4} = 4^{(\log n)/2} = 2^{(\log n)} = n$

Look-up Table for Four Russians Technique



Let $t = (\log n)/4$. Then the number of entries
In the lookup table: $4^t \times 4^t = n$

Computing the scores for each entry in the table requires dynamic programming for a $(\log n)$ by $(\log n)$ alignment: $(\log n)^2$
Altogether: $n (\log n)^2$

Four Russians Speedup Runtime

- Since computing the lookup table *Score* of size n takes $O(n (\log n)^2)$ time, the running time is mainly limited by the n^2/t^2 accesses to the lookup table
- Each access takes $O(\log n)$ time
- Overall running time: $O([n^2/t^2]*\log n)$
- Since $t = \log n$, substitute in:
- $O([n^2/\{\log n\}^2]*\log n) = O(n^2/\log n)$

So Far... (restricted to block alignment)

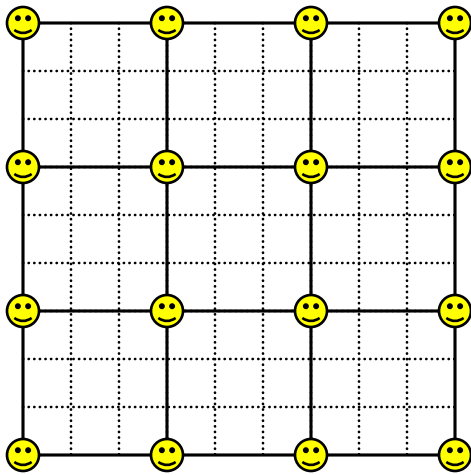
- We can divide up the grid into blocks and run dynamic programming only on the corners of these blocks
- In order to speed up the mini-alignment calculations to under n^2 , we create a lookup table of size n , which consists of all scores for all t -nucleotide pairs
- Running time goes from quadratic, $O(n^2)$, to subquadratic: $O(n^2/\log n)$

Block Alignment vs. LCS

- In block alignment, we only care about the corners of the blocks.
 - In LCS, we care about all points on the edges of the blocks, because those are points that the path can traverse.
 - Recall, each sequence is of length n , each block is of size t , so each sequence has (n/t) blocks.
-

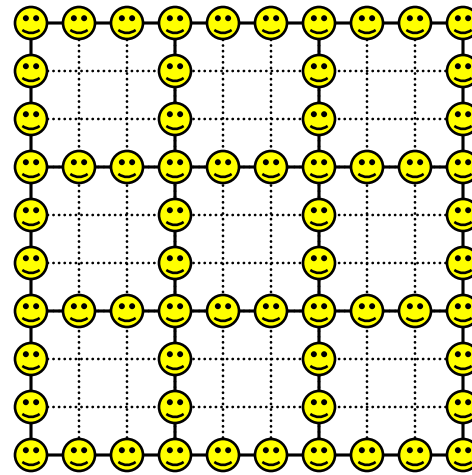
How Many Points Of Interest?

block alignment



How many blocks?
 $(n/t) * (n/t) = (n^2/t^2)$

longest common subsequence



How many points of interest? $O(n^2/t)$

n/t rows with n vertices each

n/t columns with n vertices each

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

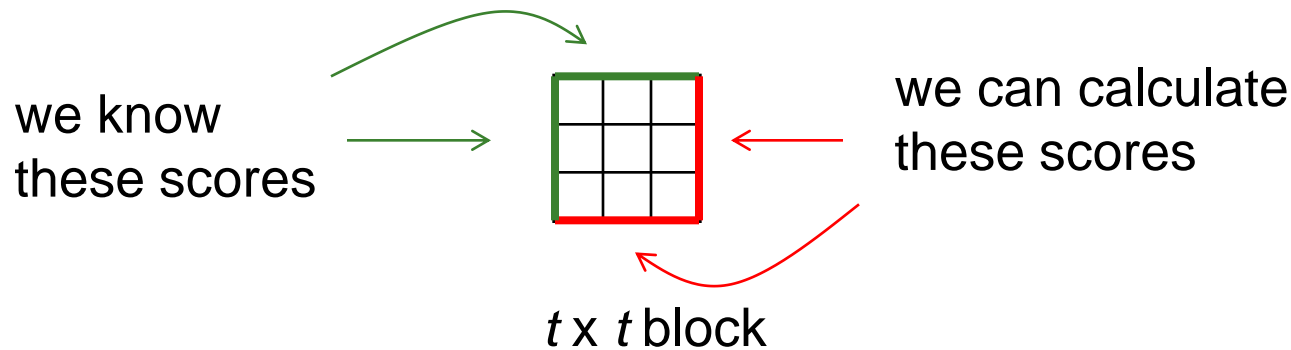
0	1	2	3	4	5	6	7	8	9
1									
2									
3									
4									
5									
6									

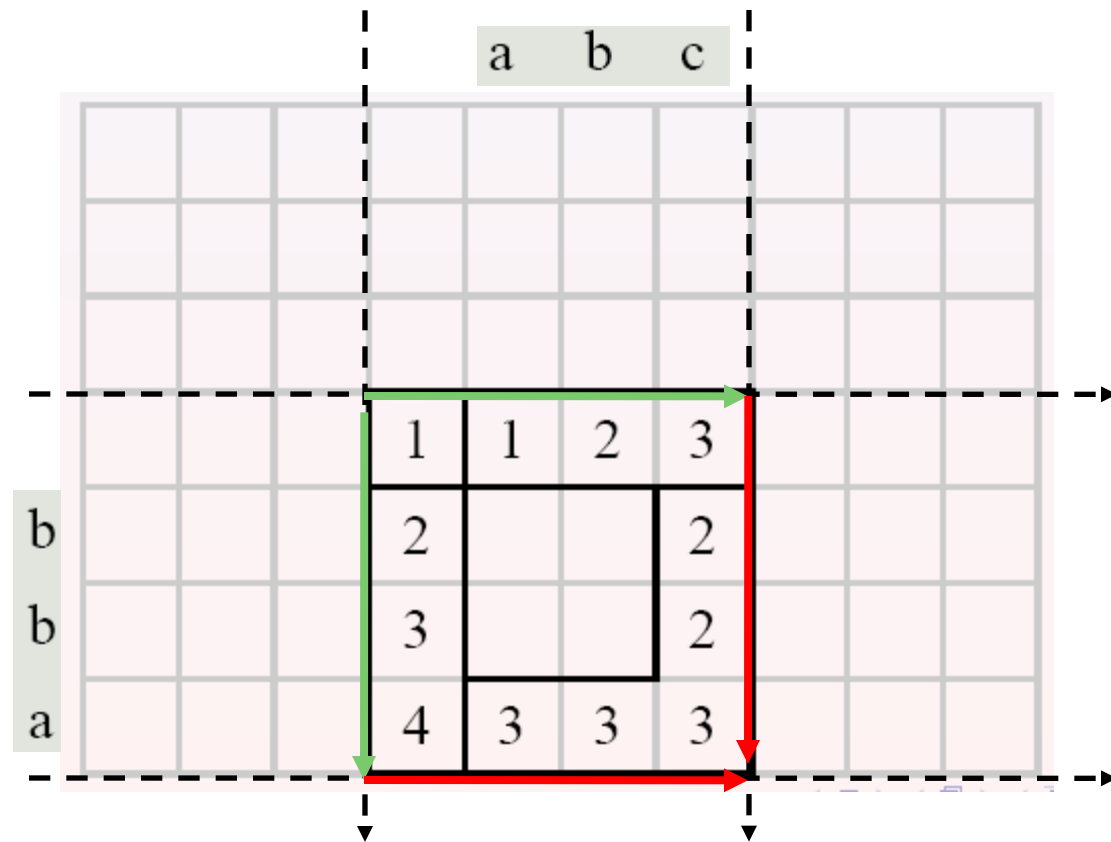
0	1	2	3	4	5	6	7	8	9
1			2						
2			1						
3	2	1	1						
4			2						
5			3						
6	5	4	4						

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

Traversing Blocks for LCS (cont'd)

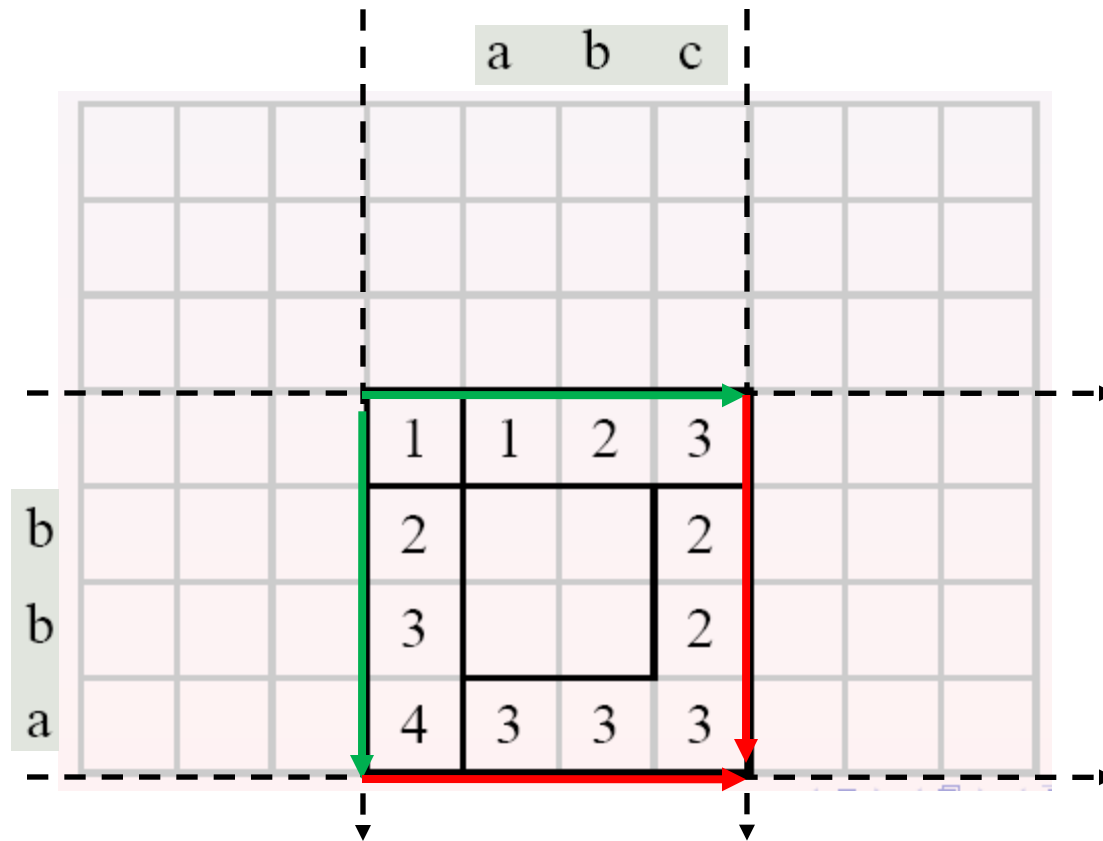
- If we used regular dynamic programming to compute the grid, it would take quadratic, $O(n^2)$ time, but we want to do better.
- Use the “Four Russians” Tabulation!





$I = ((1, 2, 3), (2, 3, 4), abc, bba)$

$O = (4, 3, 3, 3, 2, 2, 3)$



$I = (1, 1, 2, 3), (1, 2, 3, 4), abc, bba)$

$$n^t * n^t * 4^t * 4^t = (4n)^{2t}$$

This will be a huge table!
we need another trick...

$O = (4, 3, 3, 3, 2, 2, 3)$

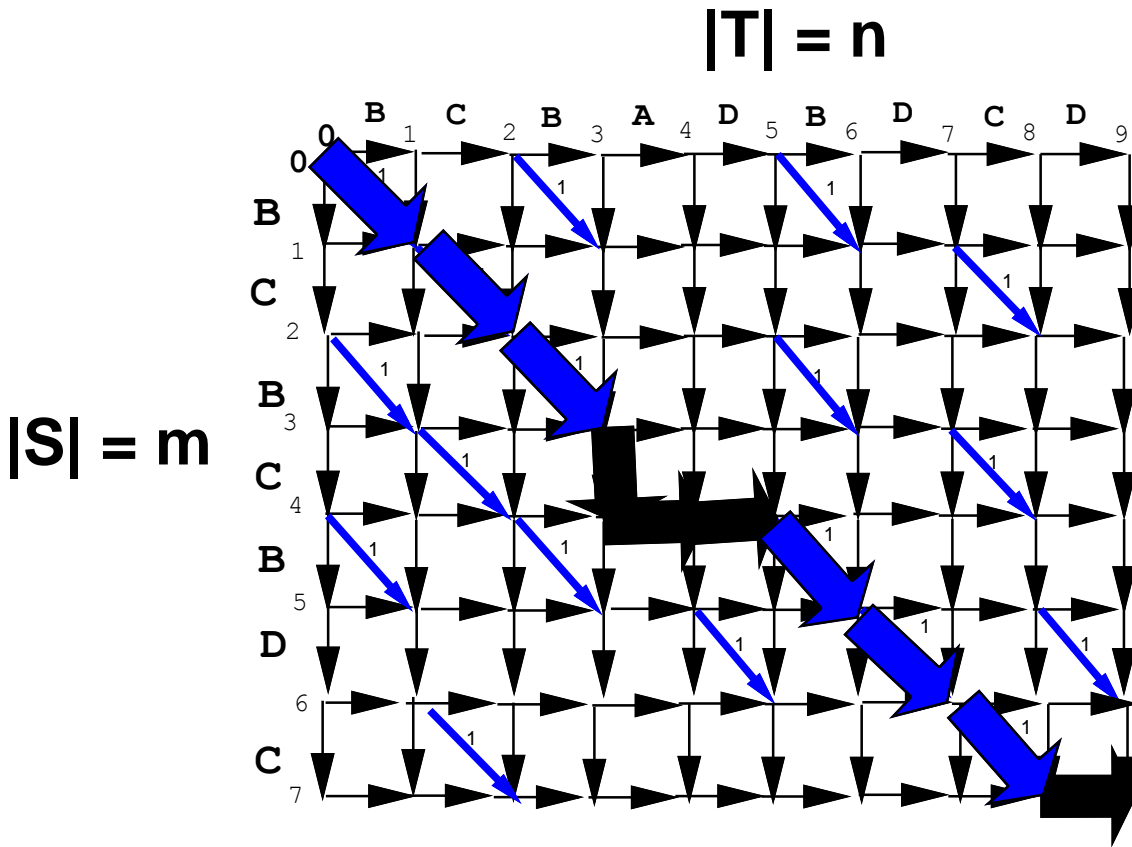
The Longest Common Subsequence

T = B C B A D B D C D
 | | | | | | |
S = A B C B D B D D

X = LCS(S,T) = BCBDBDD

L = |LCS(S,T)| = |BCBDBDD| = 7

The LCS Alignment Graph

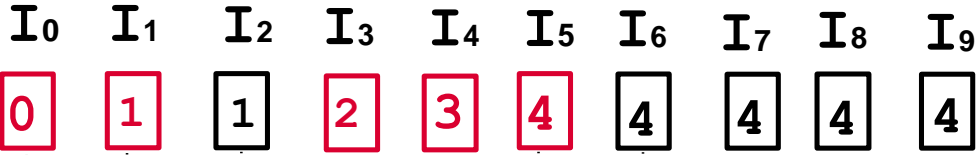
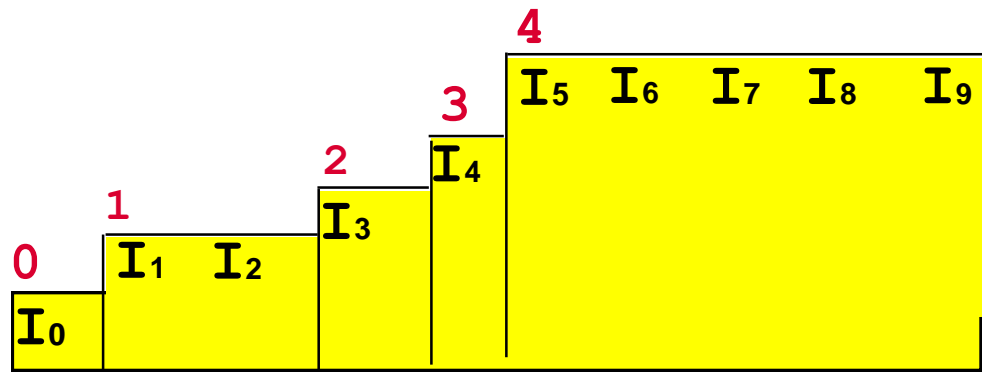


Diagonal blue arrows are match points $\{(i,j) \mid S[i] = T[j]\}$. Assigned a score of 1.

Horizontal black arrows are deletions from T . Assigned a score of 0.

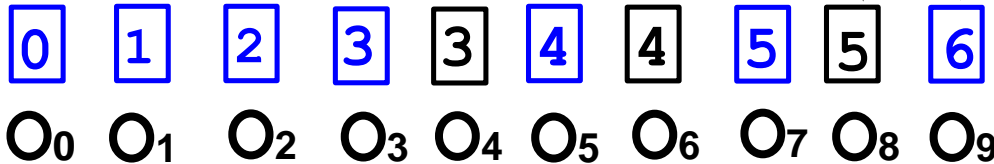
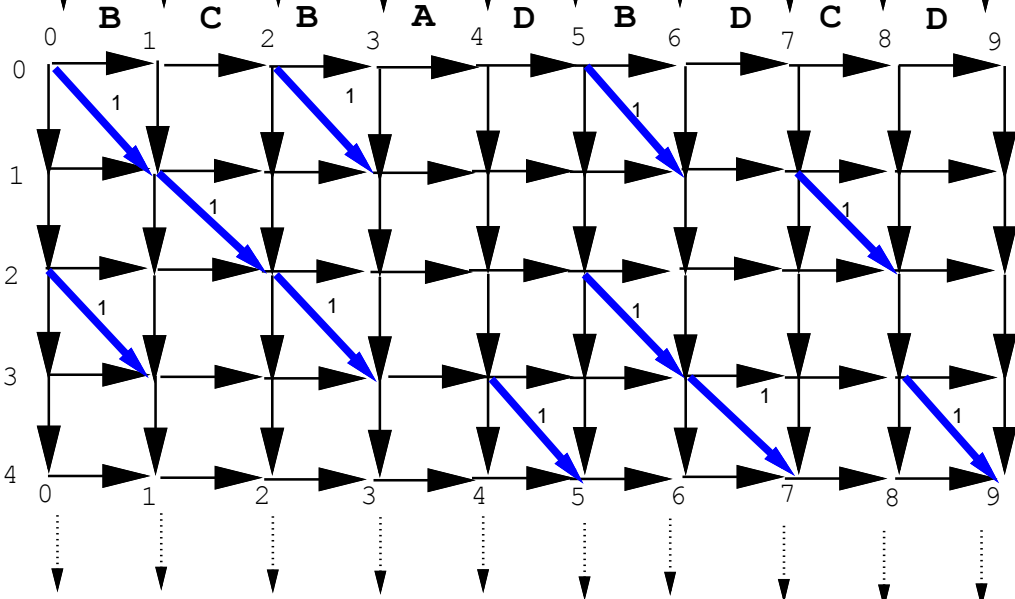
Vertical black arrows are deletions from S . Assigned a score of 0.

Classical Dynamic Programming: $O(n m)$
(Crochemore, Landau, Ziv-Ukelson $O(n m / \log m)$)



Observation. Due to the unit-step properties of LCS, both **I** and **O** are monotonically non-decreasing series, and their values go up by unit steps. [Hunt-Szymanski-77].

Input row I



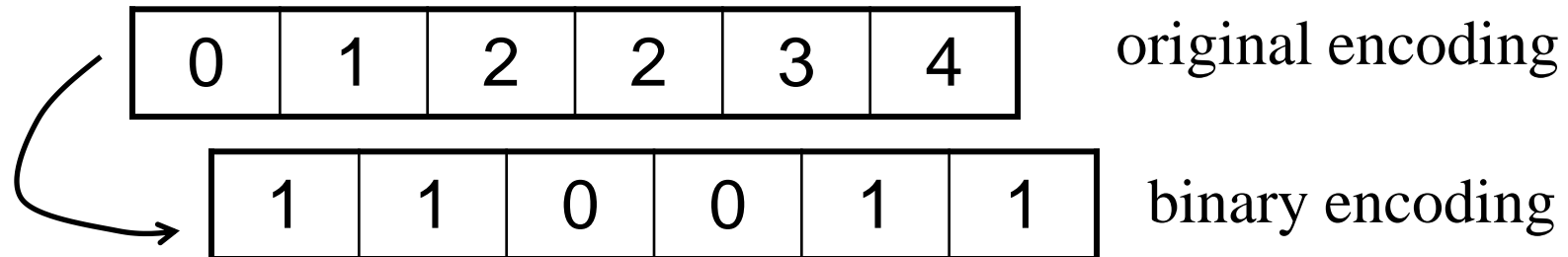
Output row O

Reducing Table Size

- Alignment scores in LCS are monotonically increasing, and adjacent elements can't differ by more than 1
- Example: 0,1,2,2,3,4 is ok; 0,1,**2,4**,5,8, is not because 2 and 4 differ by more than 1 (and so do 5 and 8)
- Therefore, we only need to store quadruples whose scores are monotonically increasing and differ by at most 1

Efficient Encoding of Alignment Scores

- Instead of recording numbers that correspond to the index in the sequences u and v , we can use binary to encode the differences between the alignment scores



	a	b	c	
b	1	1	2	3
b	2			
b	3			
a	4			

	a	b	c	
b	3	3	4	5
b	4			
b	5			
a	6			

$(1, (0, 1, 1), (1, 1, 1), abc, bba)$ $(3, (0, 1, 1), (1, 1, 1), abc, bba)$

If we have two blocks with representations (a, b, c, s, t) and (a', b, c, s, t) , then the blocks are “equivalent”:

We need to precompute only $(0,(0,1,1),(1,1,1), abc, bba)$

	a	b	c	
b	1	1	2	3
b	2	2	1	2
b	3	3	2	2
a	4	3	3	3

$(1,(0,1,1),(1,1,1),abc,bba)$

	a	b	c	
b	3	3	4	5
b	4	4	3	4
b	5	5	4	4
a	6	5	5	5

$(3,(0,1,1),(1,1,1),abc,bba)$

If we have two blocks with representations (a, b, c, s, t) and (a', b, c, s, t) , then the blocks are “equivalent”: The value of each cell in the 2nd block is equal to the value of the corresponding cell in the 1st block plus $a' - a$.

Reducing Lookup Table Size

(1,(0,1,1),(1,1,1),abc,bba)

- 2^t possible “steps” ($t =$ size of blocks)
- 4^t possible strings
 - Lookup table size is $(2^t * 2^t) * (4^t * 4^t) = 2^{6t}$
 - Computing each entry in the table: t^2
 - Total Table Construction Time: $2^{6t} t^2$
- Let $t = (\log n)/6$;
 - **Table construction time is:**
 - $2^{6((\log n)/6)} (\log n)^2 = n (\log n)^2$

Reducing Lookup Table Size

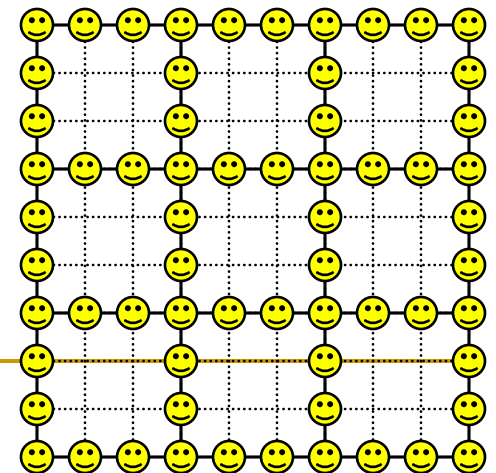
- Let $t = (\log n)/6$;

Stage 1: Table construction time is:

$$2^{6((\log n)/6)} (\log n)^2 = n (\log n)^2$$

Stage 2: alignment graph computation time is:

$$\begin{aligned} O([n^2/t^2]*t) &= O([n^2/\{\log n\}^2]*\log n) \\ &= O(n^2/\log n) \end{aligned}$$



Summary

- We take advantage of the fact that for each block of $t = O(\log n)$, we can pre-compute all possible scores and store them in a lookup table of size n , whose values can be computed in time $O(n (\log n)^2)$.
- We used the Four Russian speedup to go from a quadratic running time for LCS to subquadratic running time: $O(n^2/\log n)$