Reminder: sequence alignment in sub-quadratic time

- Last week: Sequence alignment in sub-quadratic time for unrestricted Scoring Schemes.

  1) utilize LZ78 parsing

  2) utilize Total Monotonicity property of highest scoring paths in the alignment graph. (SMAWK)

- Today: Another algorithm for sub-quadratic sequence alignment under **restricted, discrete scoring schemes**

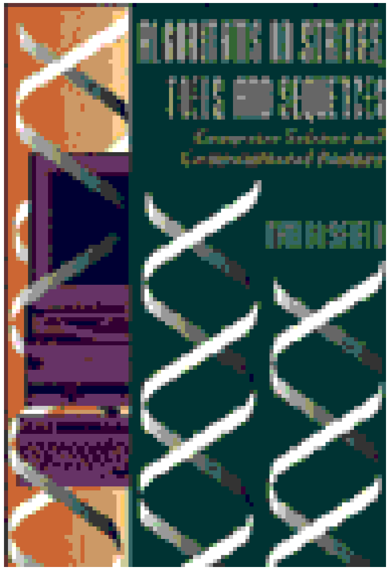
Another technique to Align Sequences in Subquadratic Time?

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

- *Another idea for exploiting repetitions: Divide the input into very small parts, pre-compute the DP for all possible values the small parts and store in a table. Then, speed up the dynamic programming via Table Lookup.*

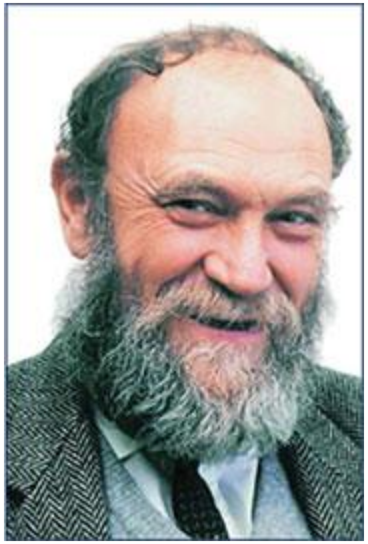
The "*Four-Russians*" *technique for speeding up for dynamic programming*

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

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

Previous Results [Masek and Paterson 1980]

- An $O(n^2 / \log n)$ time global alignment algorithm.

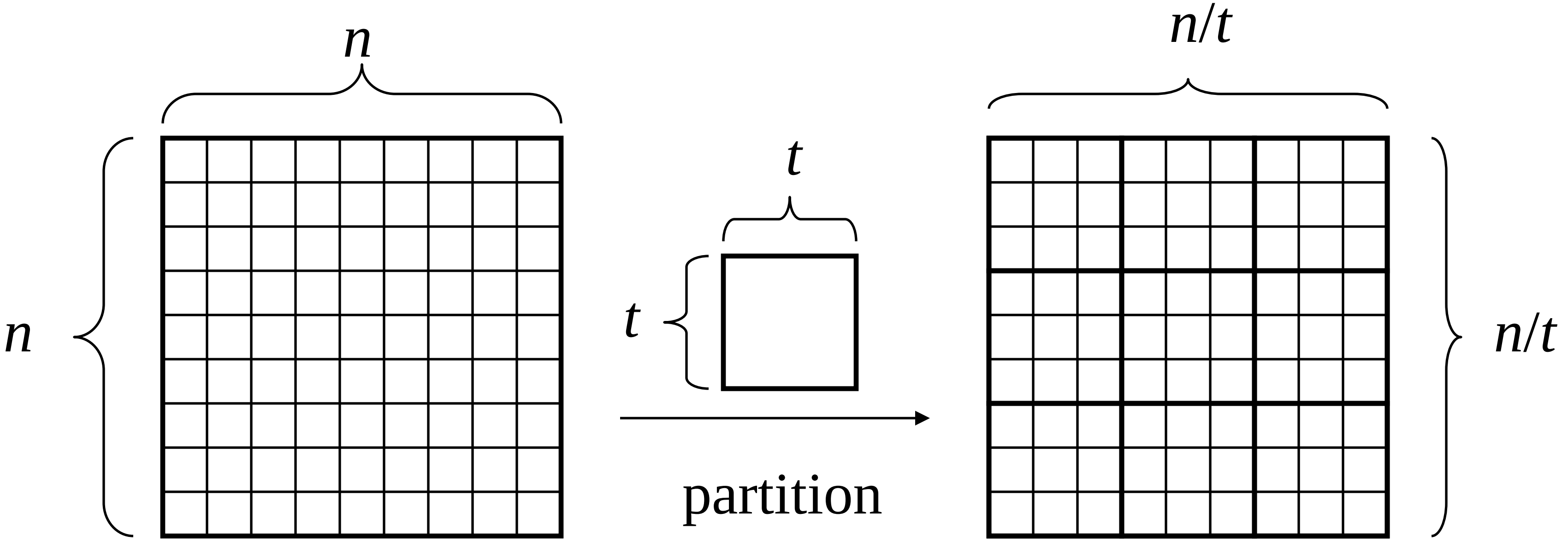
- Constant size alphabet.

- Restricted to discrete scoring schemes.

Open Problem [Masek and Paterson 1980]

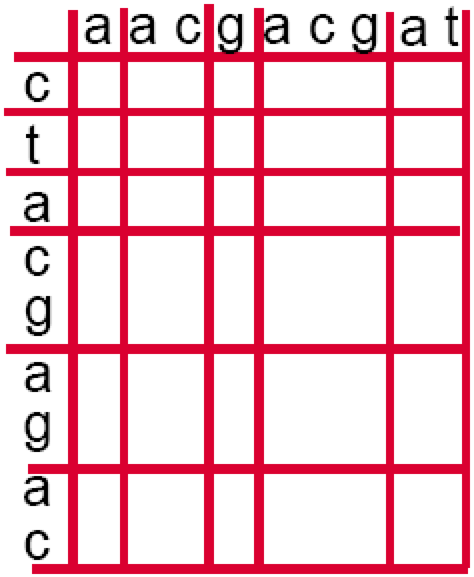
Can a better algorithm be found for the constant alphabet case, which does not restrict the scoring matrix values?

Partitioning Alignment Grid into Blocks of equal size t



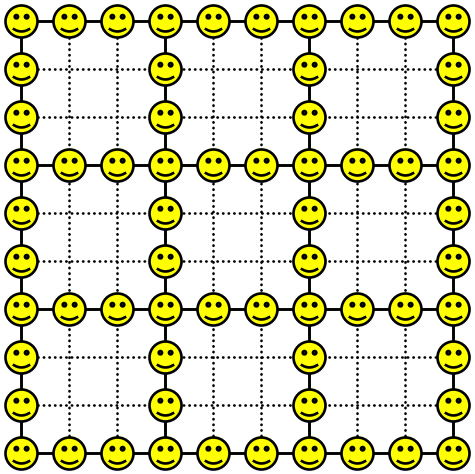
How Many Points Of Interest?

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)

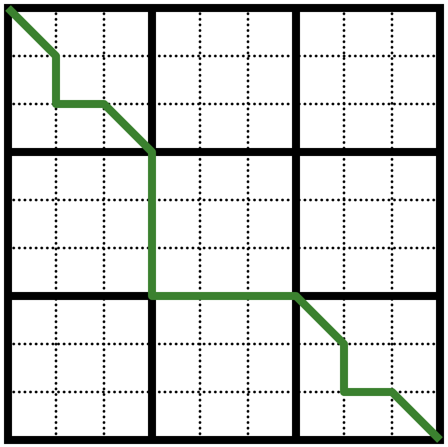
n/ t rows with n vertices each

n/ t columns with n vertices each

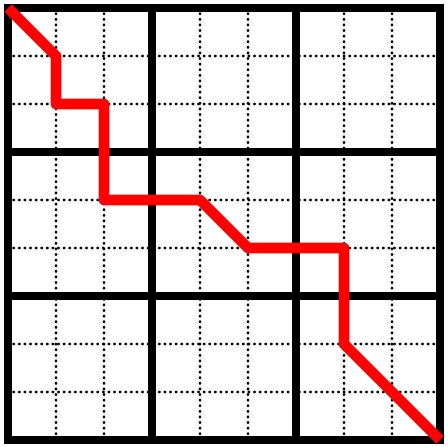
Outline

- Demonstrate the “Four Russians” technique on a simpler problem: Block Alignment.
- Extend “Four Russians” to the standard sequence alignment problem: the “tabulation explosion” challenge….
- Discuss “discrete scoring schemes” and the “unit step” property. Example of LCS.
- Four Russians algorithm for sub-quadratic sequence alignment under discrete scoring schemes

Start with a Simpler Problem: Block Alignment



valid



invalid

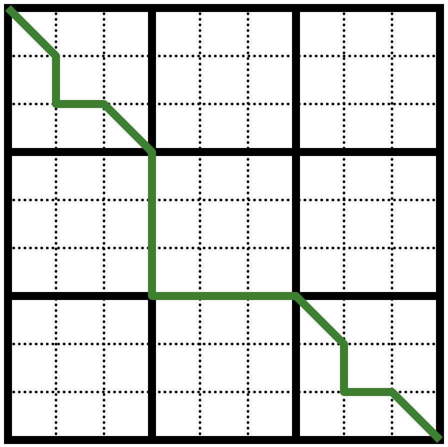
Block Alignment: legitimate operations

- **Block alignment** of sequences ***u*** and ***v:***

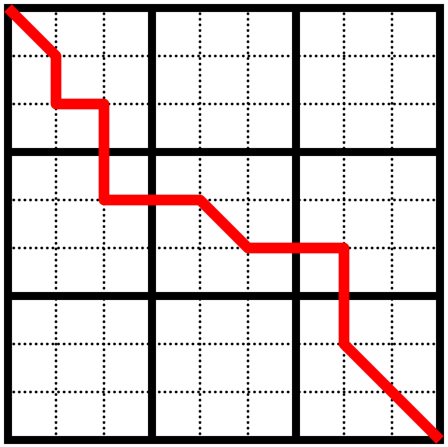
1. An entire block(i.e. substring) in ***u*** is aligned with an entire block in ***v.***
2. An entire block(substring) is inserted.
3. An entire block(substring) is deleted.

- **Block path**: a path that traverses every *t* x *t* square through its corners

Block Alignment: Examples



valid

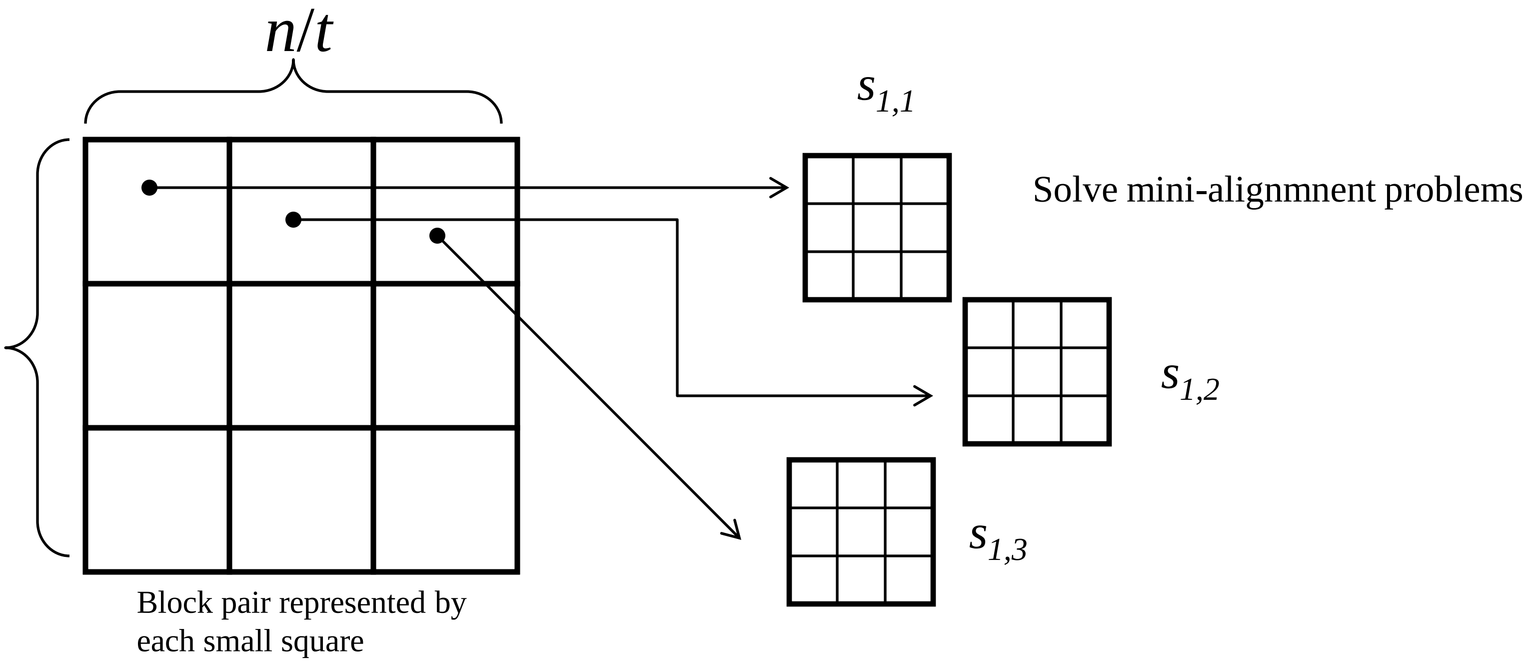


invalid

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 x n edit graph partitioned into t x t subgrids
- Output: The **block alignment** of ***u*** and ***v*** with the maximum score (longest **block path** through the edit graph)
- How do we solve this in two-stages by partitioning to t by t blocks?

Stage 1: compute the mini-alignments



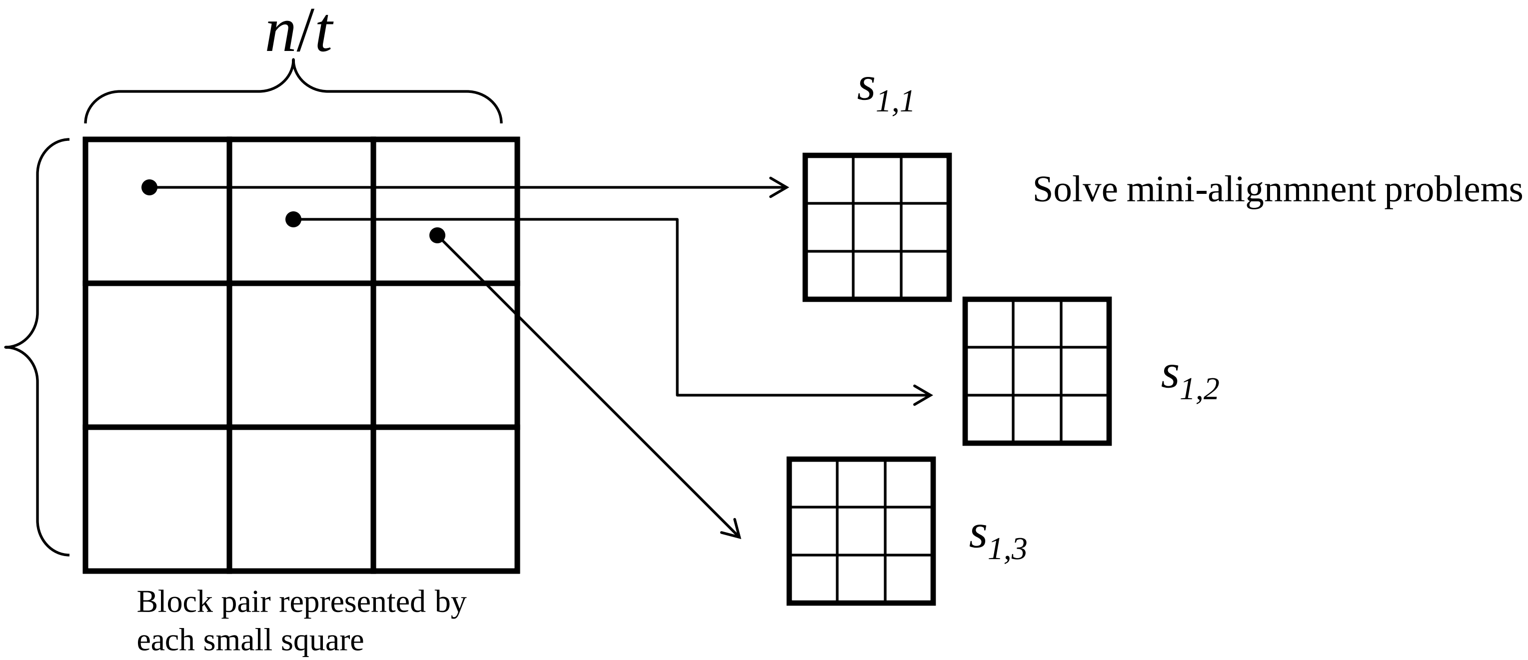
Constructing Alignments within Blocks

- To solve: compute alignment score $ß_{i,j}$ for each pair of blocks $|u_{(i-1)*t+1}\ldots u_{i*t}|$ and $|v_{(j-1)*t+1}\ldots v_{j*t}|$
- How many blocks are there per sequence?

  (n/t) blocks of size t
- How many pairs of blocks for aligning the two sequences?

  (n/t) x (n/t)
- For each block pair, solve a mini-alignment problem of size t x t

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 ***u*** and first *j* blocks of ***v***

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma_{\text{block}} \\ s_{i,j-1} - \sigma_{\text{block}} \\ s_{i-1,j-1} + \beta_{i,j} \end{cases}$$

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)

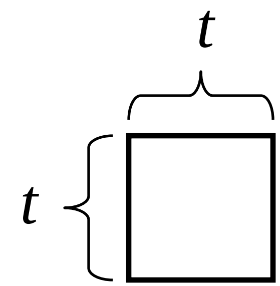
- Computing all $\beta_{i,j}$ requires solving

  $(n/t)*(n/t) = n^2/t^2$ 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? (utilize repetitive mini-blocks…)

Four Russians Technique

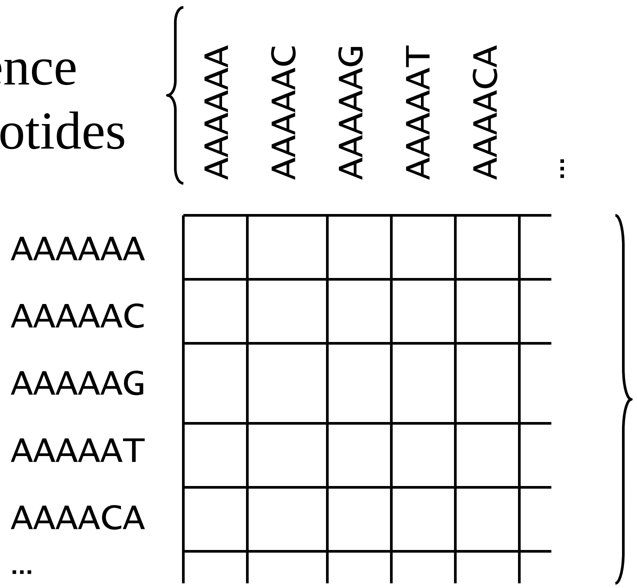


- Let $t = \log(n)$, where t is block size, n is sequence size.
- Instead of having $(n/t)*(n/t)\ ) = n^2/t^2$ mini-alignments, construct $4^t \times 4^t$ mini-alignments for all pairs of strings of t nucleotides (huge size), and put in a lookup table.
- However, size of lookup table is not really that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

each sequence has *t* nucleotides

Lookup table "*Score*"



size is only *n*, instead of (*n*/*t*)*(*n*/*t*)

Let t = (log<u>*n*</u>)/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: $(\log n)^2$

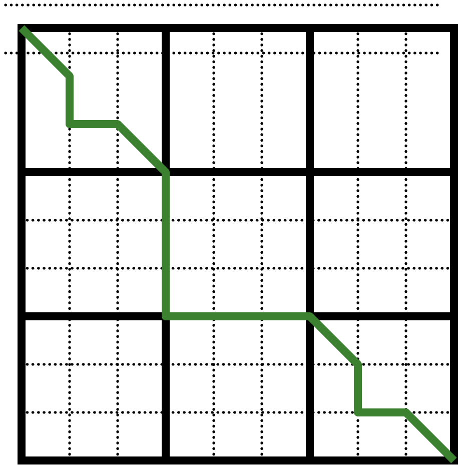
Altogether: $n (\log n)^2$ **(instead of** $O(n^2)$**)…)**

New Recurrence

- The new lookup table *Score* is indexed by a pair of *t*-nucleotide strings, so

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma_{\text{block}} \\ s_{i,j-1} - \sigma_{\text{block}} \\ s_{i-1,j-1} + Score(i^{\text{th}} \text{ block of } \boldsymbol{v},\ j^{\text{th}} \text{ block of } \boldsymbol{u}) \end{cases}$$

O(log*n*) time



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… (restriced to block alignment)

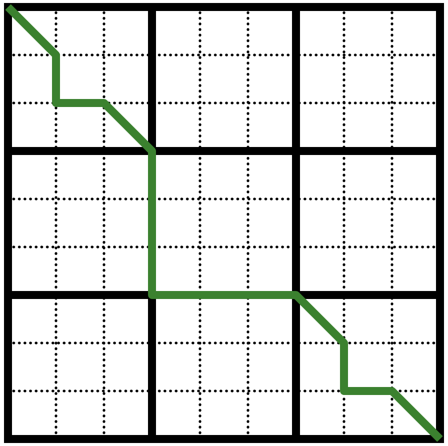
- We can divide up the grid into blocks and run dynamic programming only on the corners of these blocks
- In order to speed up the mini-alignment calculations to under n^2, we create a lookup table of size n, which consists of all scores for all t-nucleotide pairs
- Running time goes from quadratic,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

Outline

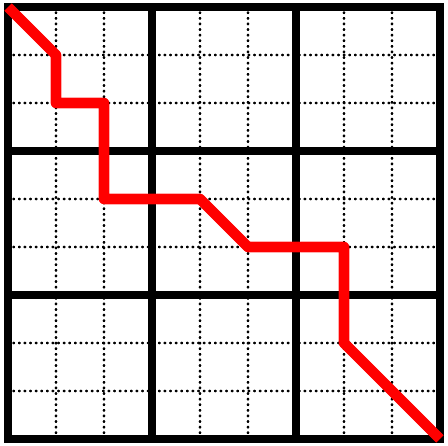
- Demonstrate the “Four Russians” technique on a simpler problem: Block Alignment.
- **Extend “Four Rusians” to the standard sequence alignment problem: the “tabulation explosion” challenge….**
- Discuss “discrete scoring schemes” and the “unit step” properties of scores for neighboring cells in the DP table for these schemes. Example of LCS.
- Four Russians algorithm for sub-quadratic sequence alignment under discrete scoring schemes

Four Russians Speedup for LCS

- Unlike the block partitioned graph, the LCS path does not have to pass through the vertices of the blocks.



block alignment



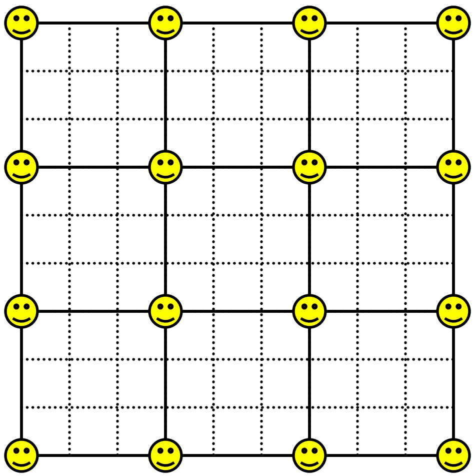
longest common subsequence

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?

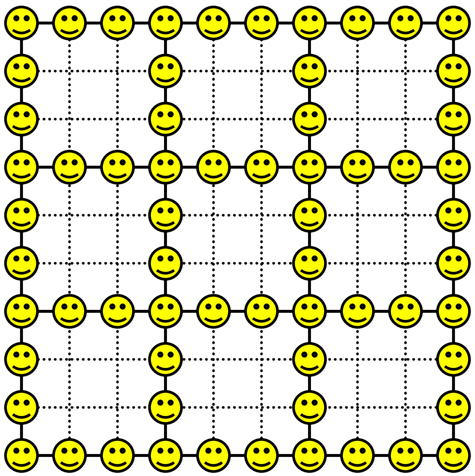
block alignment



How may 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									

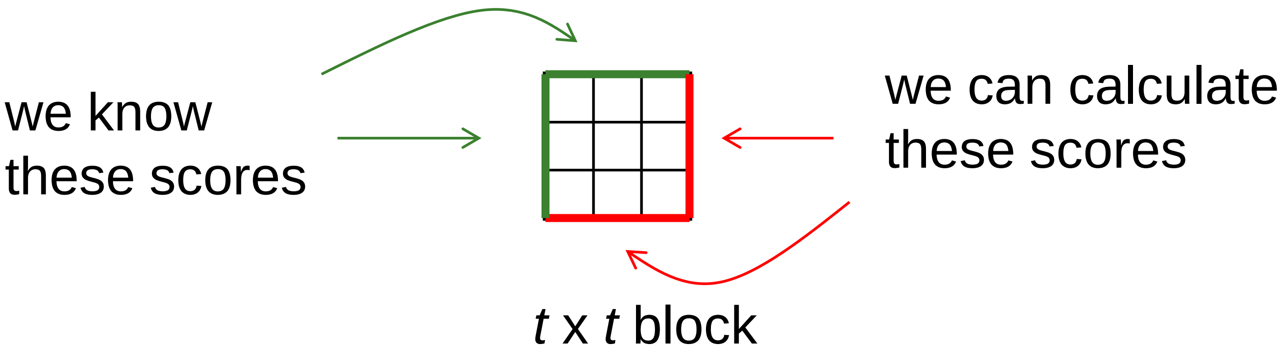
0	1	2	3	4	5	6	7	8	9
1			2						
2			1						
3	2	1	1						
4									
5									
6									

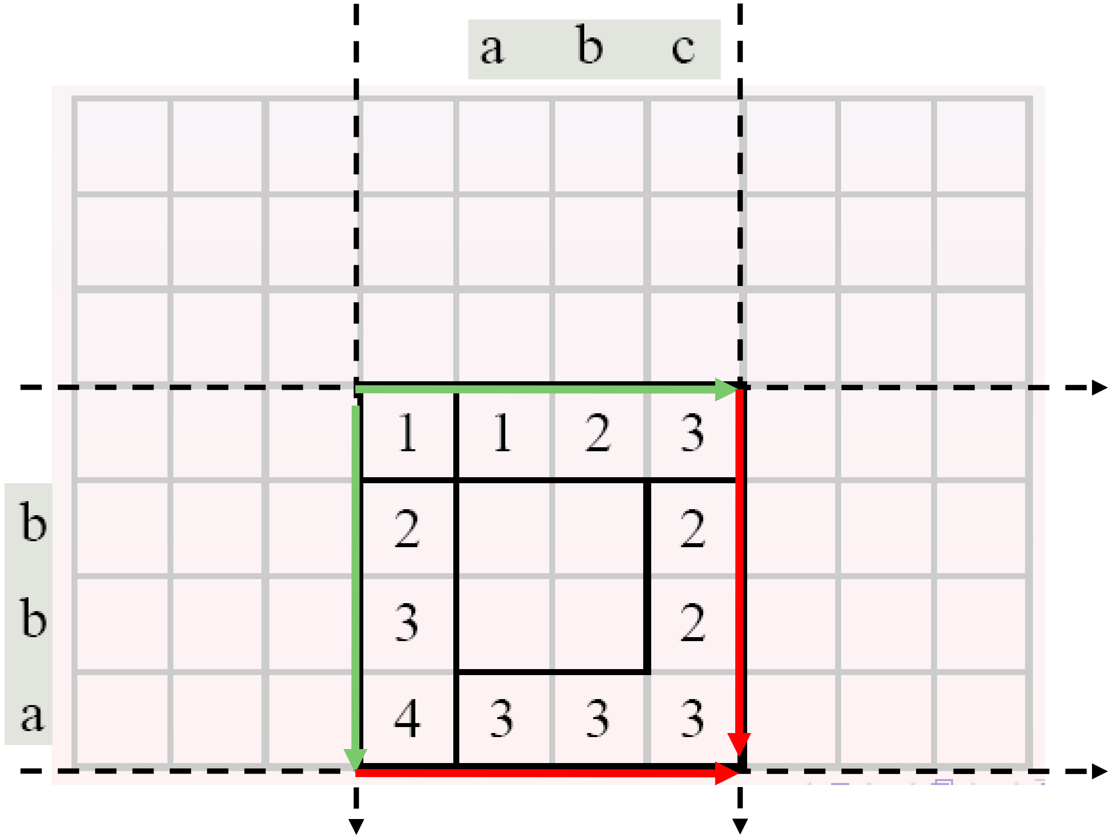
0	1	2	3	4	5	6	7	8	9
1			2						
2			1						
3	2	1	1						
4			2						
5			3						
6	5	4	4						

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, 1, 2, 3), (1, 2, 3, 4), abc, bba)

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

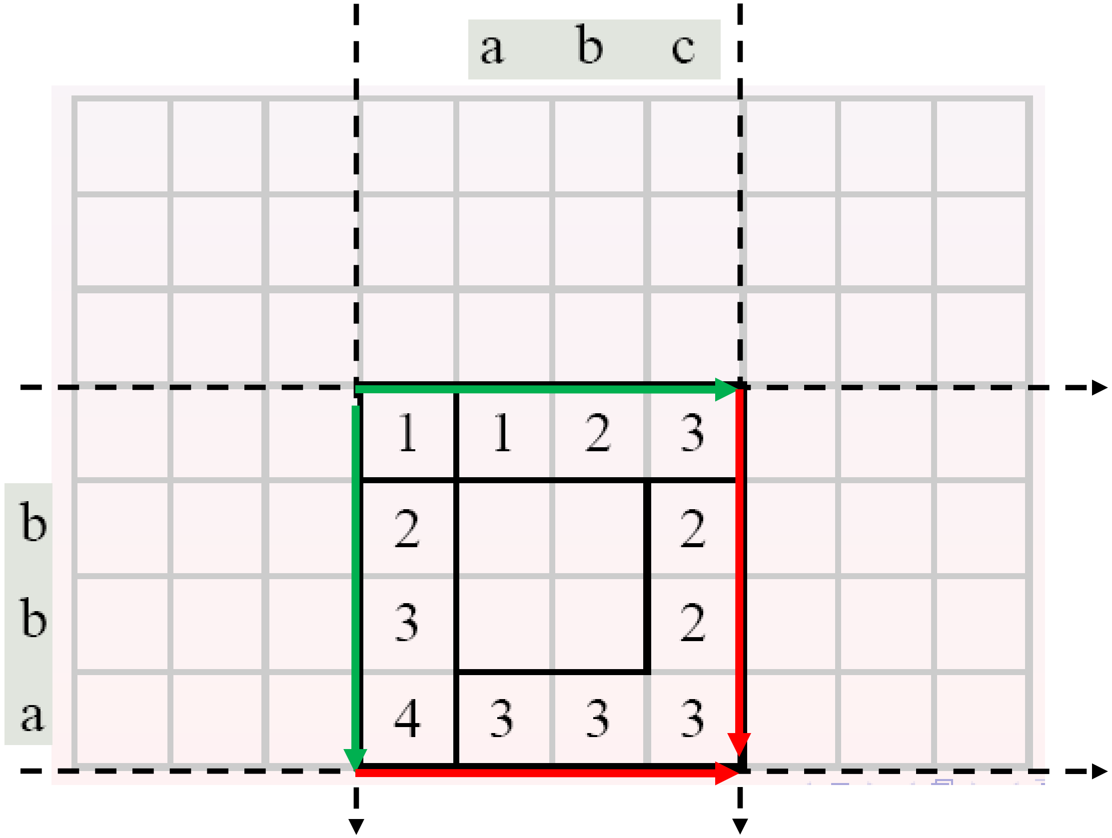
Traversing Blocks for LCS

- **New Problem:** Given alignment scores $s_{i,*}$ in the first row and scores $s_{*,j}$ in the first column of a t x t mini square, compute alignment scores in the last row and column of the minisquare.
- To compute the last row and the last column score, we use these 5 variables:
- 1. value in upper left cell.

1. alignment scores $s_{i,*}$ in the first row
2. alignment scores $s_{*,j}$ in the first column
3. substring of sequence *u* in this block (4^t possibilities)
4. substring of sequence *v* in this block (4^t possibilities)

Four Russians Speedup

- Build a lookup table for all possible values of the four variables:
  1. all possible scores for the first row $s_{*,j}$
  2. all possible scores for the first column $s_{*,j}$
  3. substring of sequence u in this block (4^t possibilities)
  4. substring of sequence v in this block (4^t possibilities)
- For each quadruple we store the value of the score for the last row and last column.



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

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

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

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

Outline

- Demonstrate the “Four Russians” technique on a simpler problem: Block Alignment.
- Extend to the standard sequence alignment problem: the “tabulation explosion” challenge….
- **Discuss “discrete scoring schemes” and the “unit step” propertty.**
- Four Russians algorithm for sub-quadratic sequence alignment under discrete scoring schemes

The Longest Common Subsequence

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

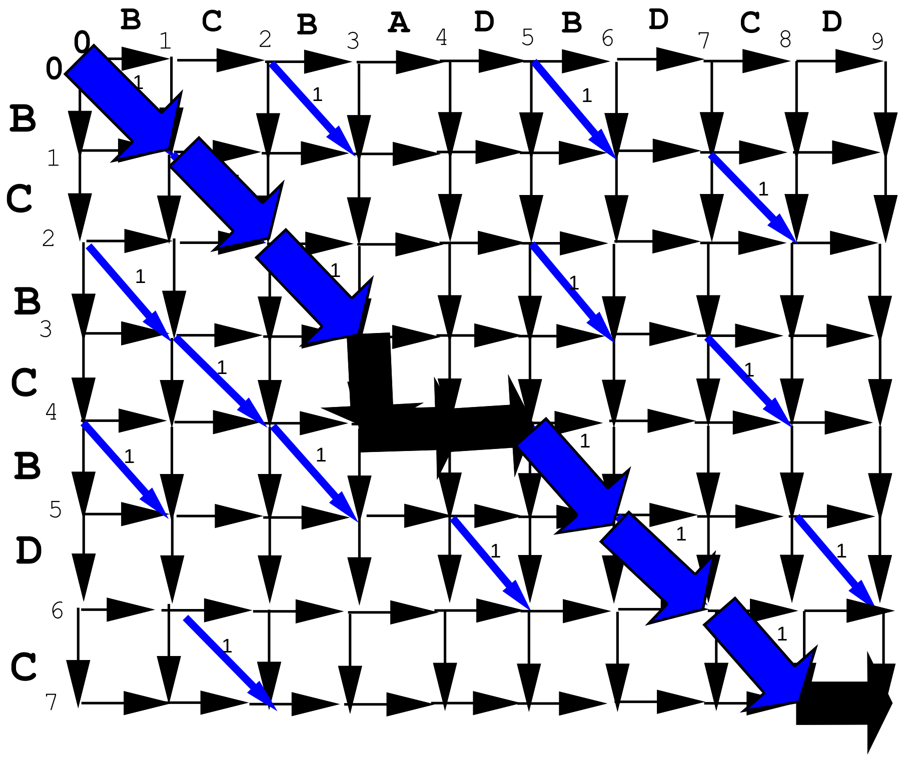
X = LCS(S,T) = BCBDBDD

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

The LCS Alignment Graph

|T| = n

|S| = m



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

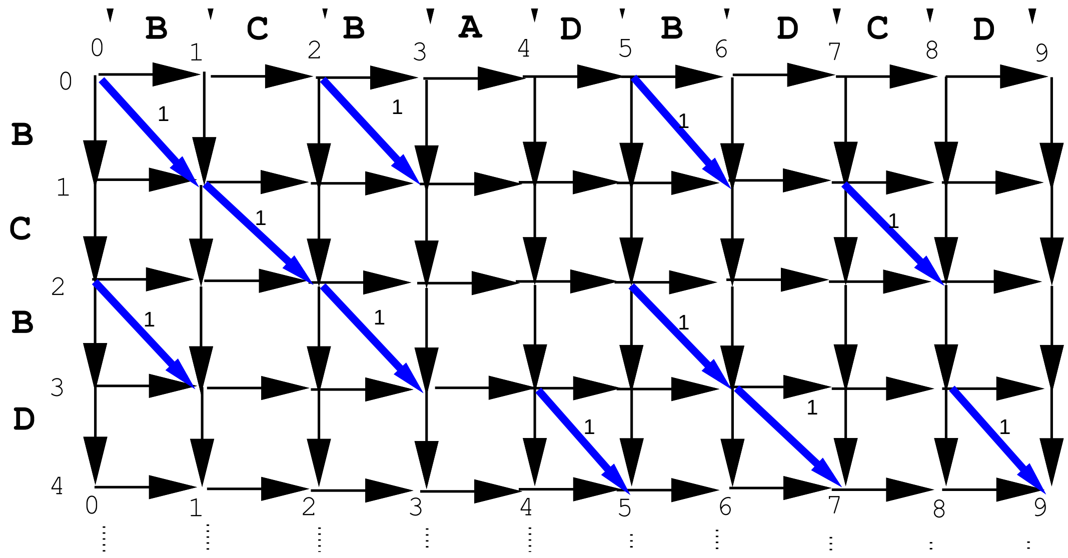
Horizontal black arrows are deletions from T. Assigned a score of 0.

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))

Theorem (Hunt-Szymanski 77)
Alignment scores in LCS are monotonically increasing, and **adjacent elements can't differ by more than 1**

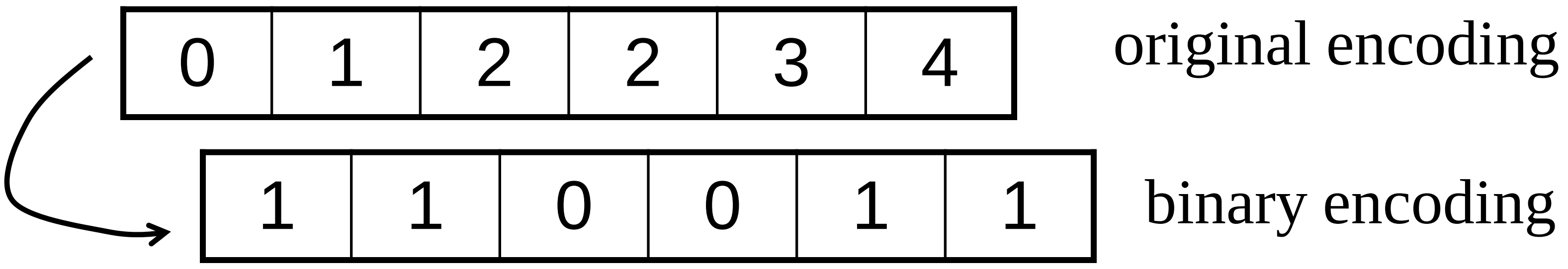


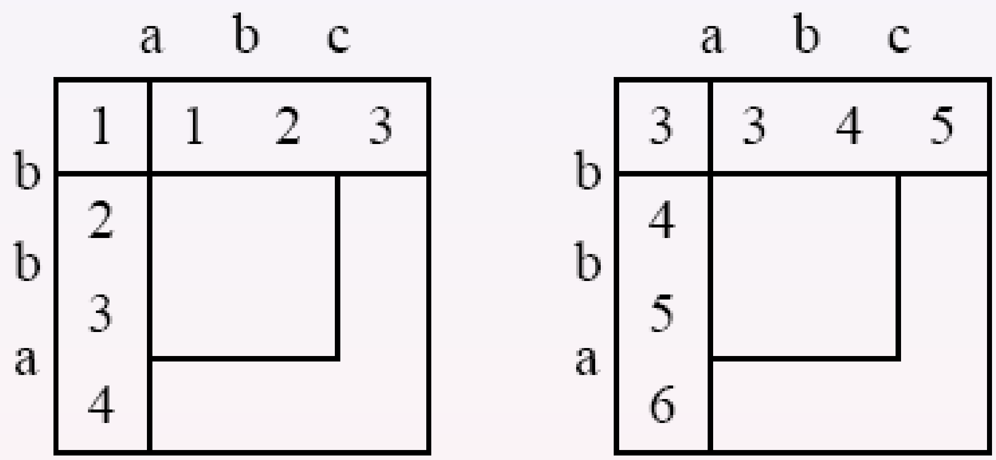
Reducing Table Size

- 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

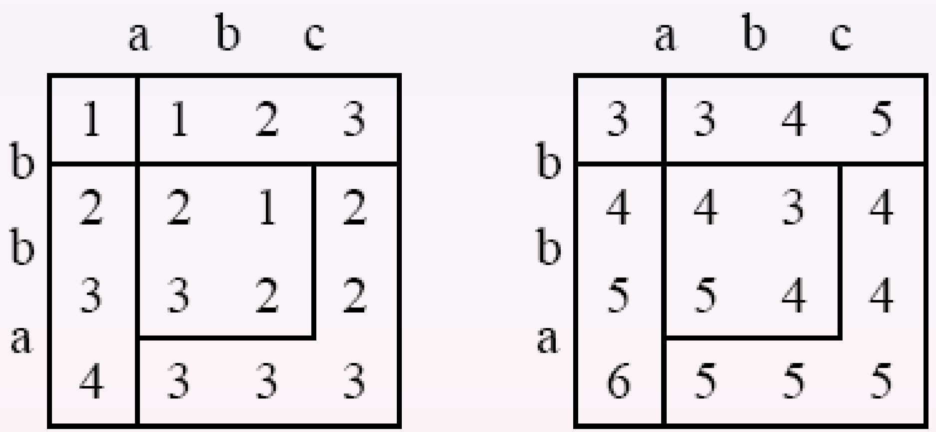




(1,(0,1,1),(1,1,1),abc,bba) (3,(0,1,1),(1,1,1),abc,bba)

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)



(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": 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 $t = (\log n)/6$;

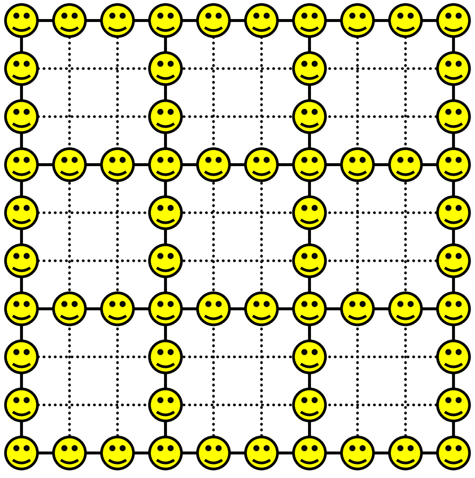
  Stage 1: Table construction time is:

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

  Stage 2: alignment graph computation time is:

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

  $$=O(n^2/\log n)$$



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* $O(n (\log \underline{n})^2)$.
- We used the Four Russian speedup to go from a quadratic running time for LCS to subquadratic running time: $O(n^2/\log \underline{n})$