Dynamic Programming

Part II

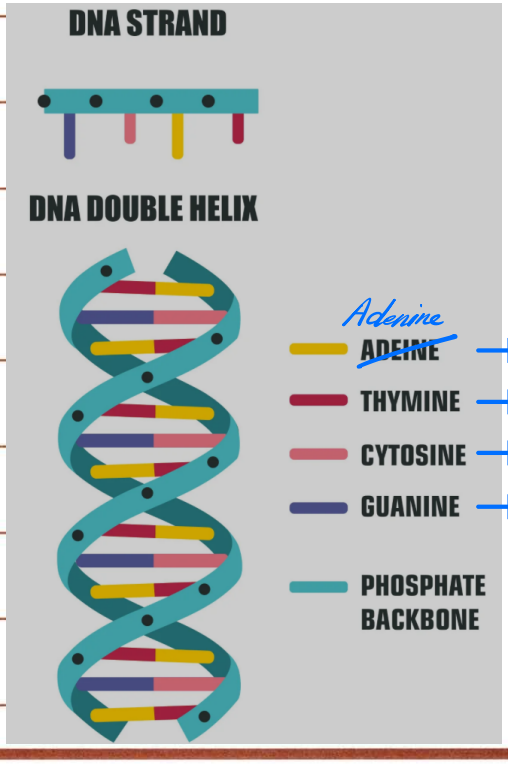
Reminder:

General Approach to Solving Optimization Problems Using Dynamic Programming

1. Characterize the structure of an opt. solution
2. Recursively define the value of an opt. solution
3. Compute the value of an opt. solution in a bottom up fashion
4. Construct an opt. solution from computed information

Sequence Alignment Problem

Background:
- A DNA strand consists of molecules called bases.



Ex.

S_1 = ACCGGTCG

S_2 = CCAGGTGGC

One possible alignment will be:

A C C _ G G T C G _

_ C C A G G T G G C

3 Gaps

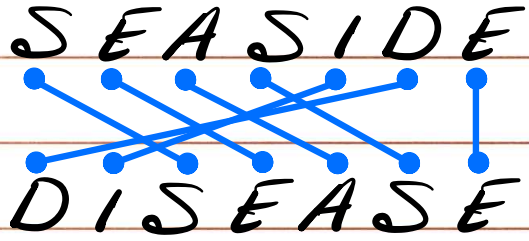
1 mismatch

We need to come up with a concise definition of similarity between two strings.

Def. Given 2 strings X and Y,

$$X = \{x_1, x_2 \dots x_m\}$$
$$Y = \{y_1, y_2 \dots y_n\},$$

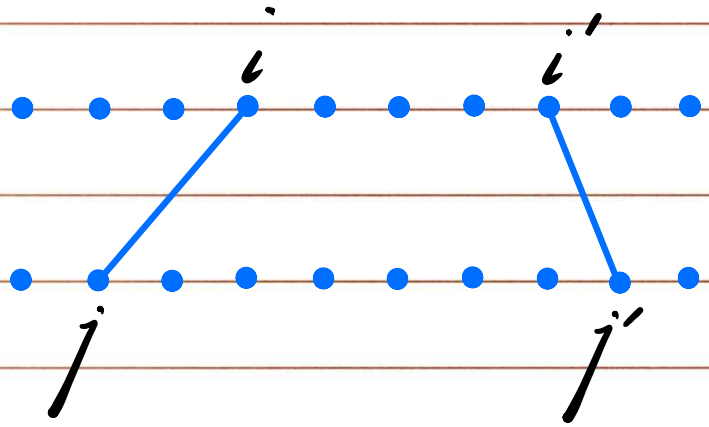
a matching is a set of ordered pairs with property that each item occurs at most once.



We have a perfect matching, but does this indicate the two strings are similar?

NO!

Def. A matching is an alignment if there are no crossing pairs.



In other words, in an alignment M, if $(i,j), (i',j') \in M$, and $i < i'$, then $j < j'$.

Cost of an alignment

For an alignment M between X and Y,

1- We incur a gap penalty of δ for each gap.

2- For each mismatch (of letters p and q) we incur a mismatch cost of α_{pq}.

	A	C	G	T
A	0	X	X	X
C		0	X	X
G			0	X
T				0

mismatch cost matrix

Def. Similarity between strings X and Y, is the minimum cost of an alignment between X and Y.

$X = \{x_1 x_2 \dots x_m\}$

$Y = \{y_1 y_2 \dots y_n\}$

Say M is an opt. solution. Then either $(x_m, y_n) \in M$ or $(x_m, y_n) \notin M$

Define $OPT(i,j)$ as the minimum cost of an alignment between $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_j$

In an optimal alignment M, at least one of the following is true:

1- $(x_m, y_n) \in M$, then

$$OPT(m,n) = OPT(m-1, n-1) + \alpha_{x_m y_n}$$

2- x_m is not-matched, then

$$OPT(m,n) = OPT(m-1, n) + \delta$$

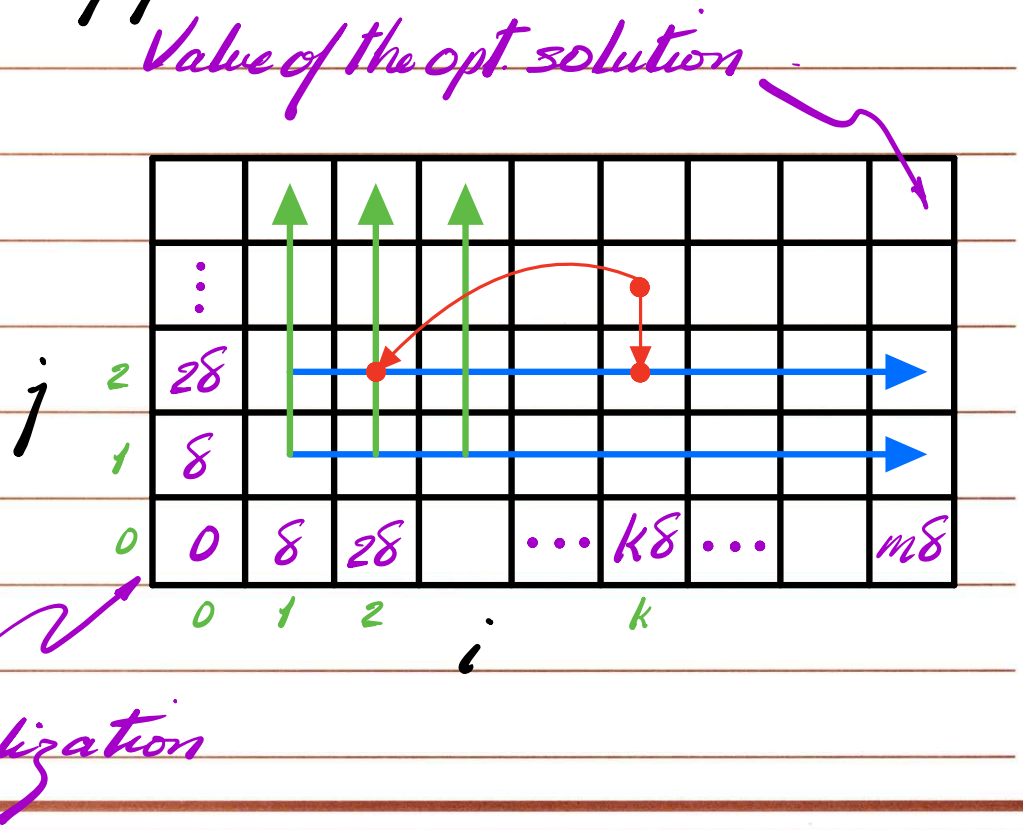
3- y_n is not-matched, then

$$OPT(m,n) = OPT(m, n-1) + \delta$$

Recurrence formula 1:

$$OPT(i,j) = \min \left[\begin{array}{l} OPT(i-1, j-1) + \alpha_{x_i y_j}, \\ OPT(i-1, j) + \delta, \\ OPT(i, j-1) + \delta \end{array} \right]$$

Bottom up pass:



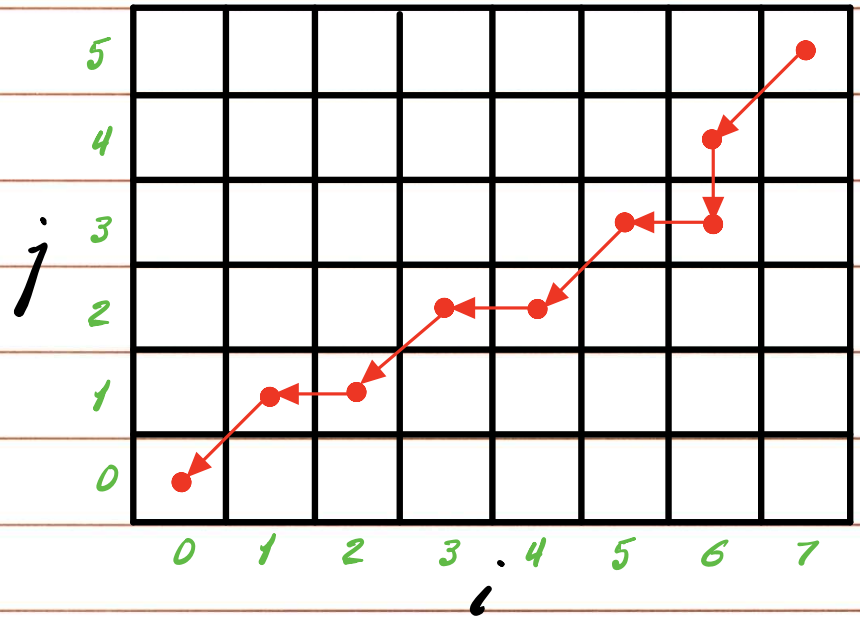
Initialize column 0 and row 0 as shown above.

```
for i = 1 to m
    for j = 1 to n
        Use recurrence formula 1
    endfor
endfor
```

Run time complexity: $O(mn)$

Is this an efficient solution? Yes!

Top-down pass:



The above example top-down pass corresponds to the following alignment:

$x_1\ x_2\ x_3\ x_4\ x_5\ x_6 - x_7$

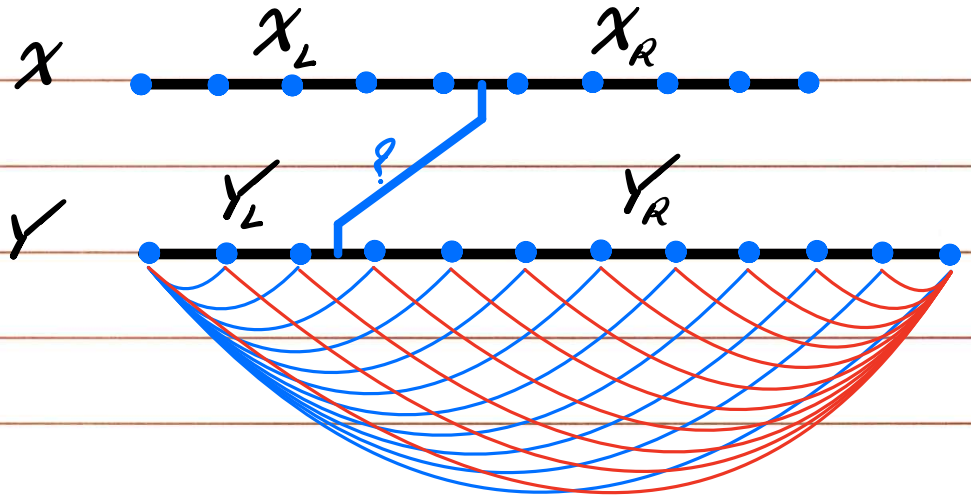
$y_1 - y_2 - y_3 - y_4\ y_5$

Memory Efficient Solution

The high level solution is based on divide and conquer.

Divide Step:

- Split X in half.
- But where is the optimal split point in Y?

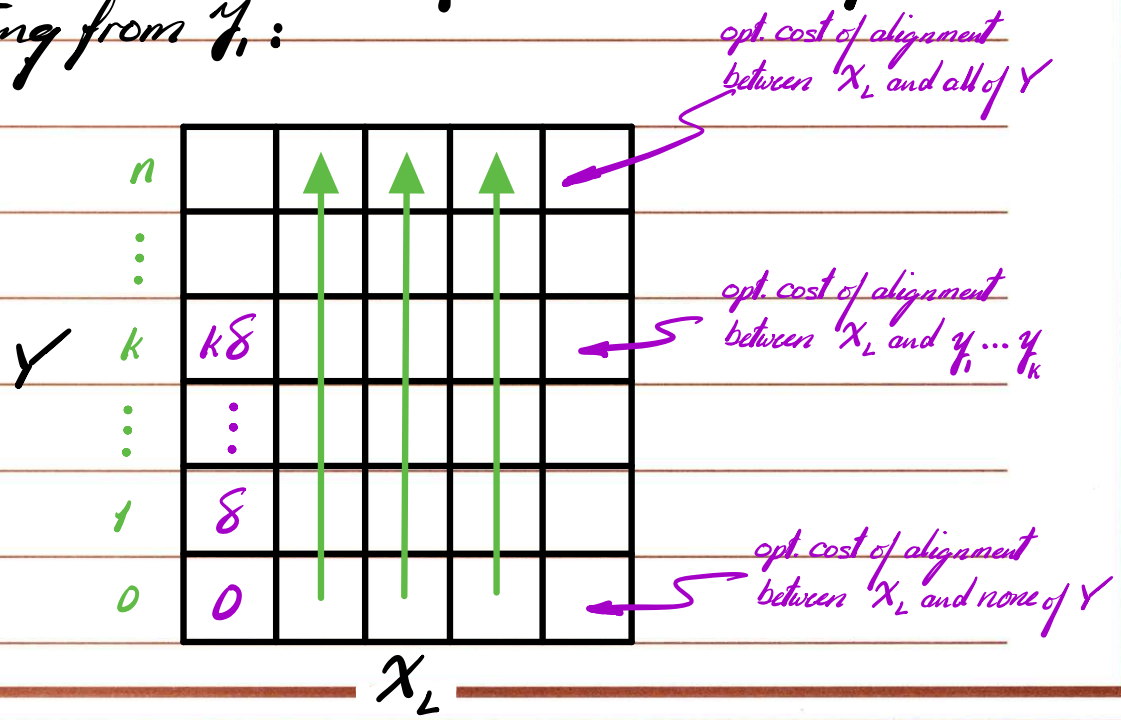


All substrings of Y starting from Y_1

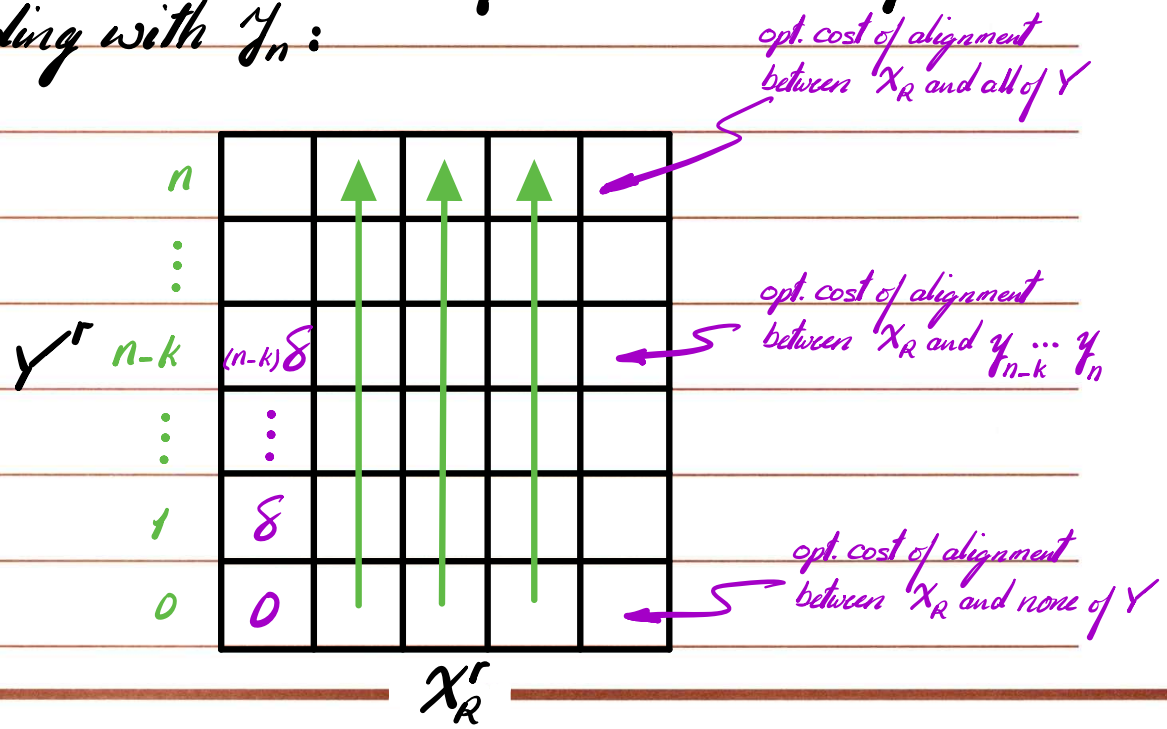
All substrings of Y ending with Y_n

We will find the optimal split point in Y using dynamic programming.

To compute the optimal cost of the alignments between X_R and each of the substrings of Y strating from y_1:



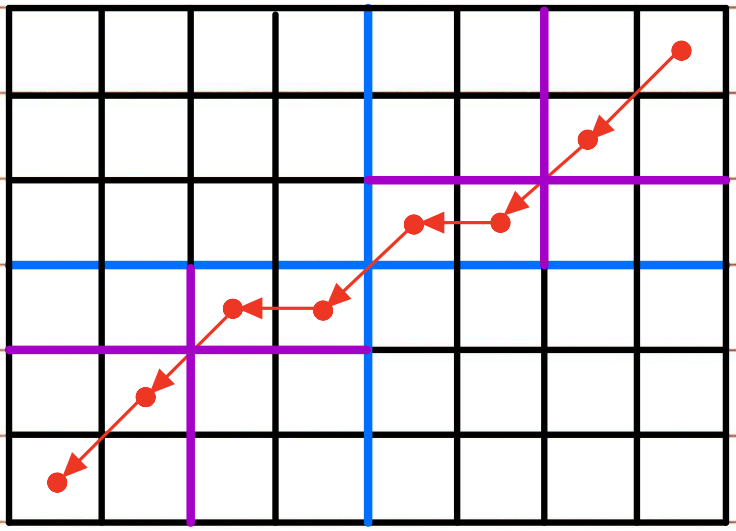
To compute the optimal cost of the alignments between X_L and each of the substrings of Y ending with y_n:



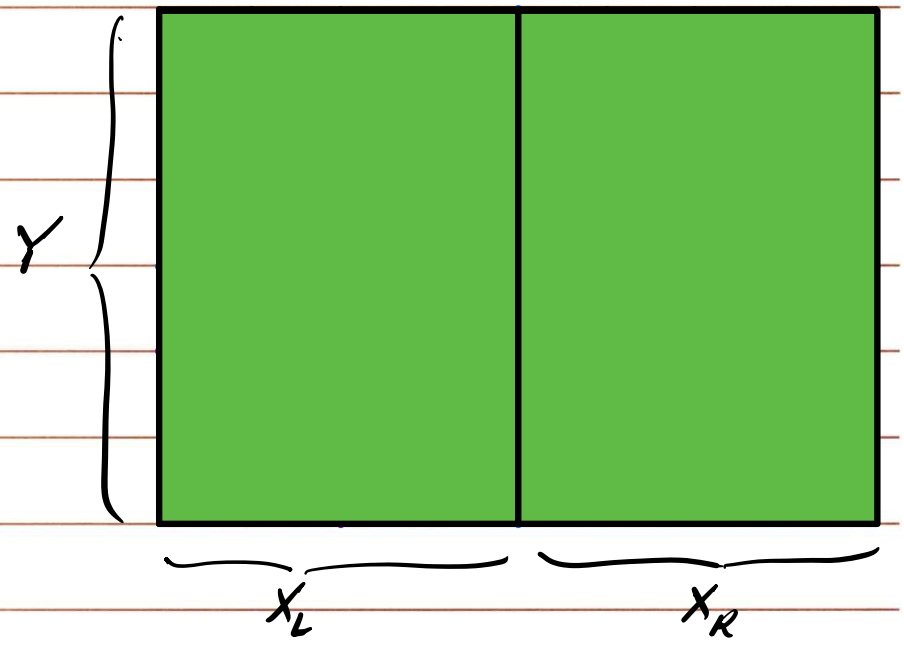
We can now compute the total opt. cost of an alignment if Y where split into $y_1 \dots y_k$ and $y_{n-k} \dots y_n$ for all $0 \leq k \leq n$, and choose the split point in Y that minimizes this total cost.

This completes the divide step.

Complexity Analysis

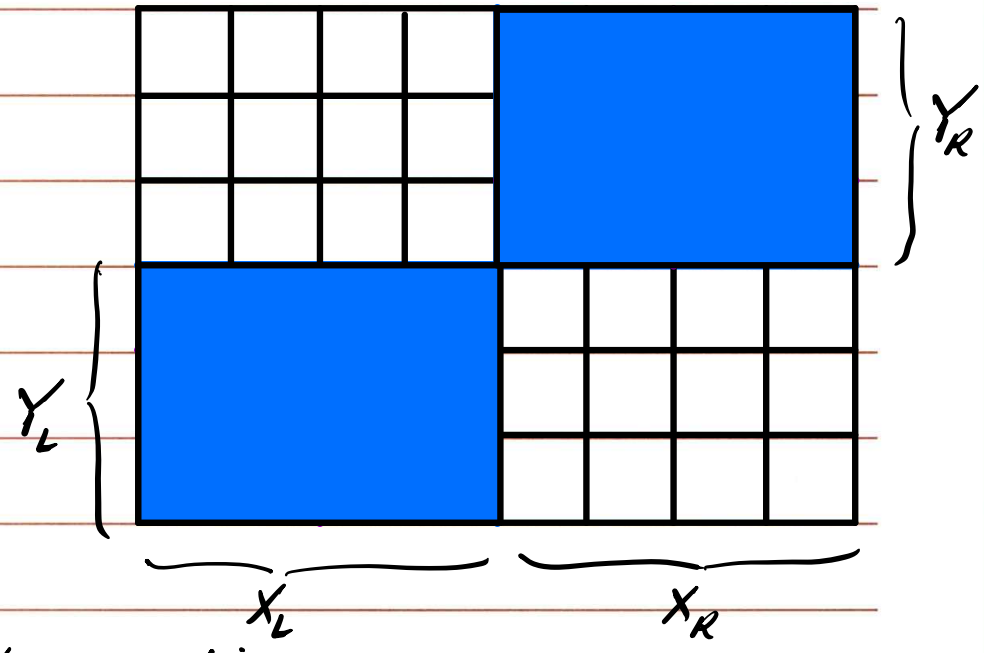


No. of operations at the root level (level 1)



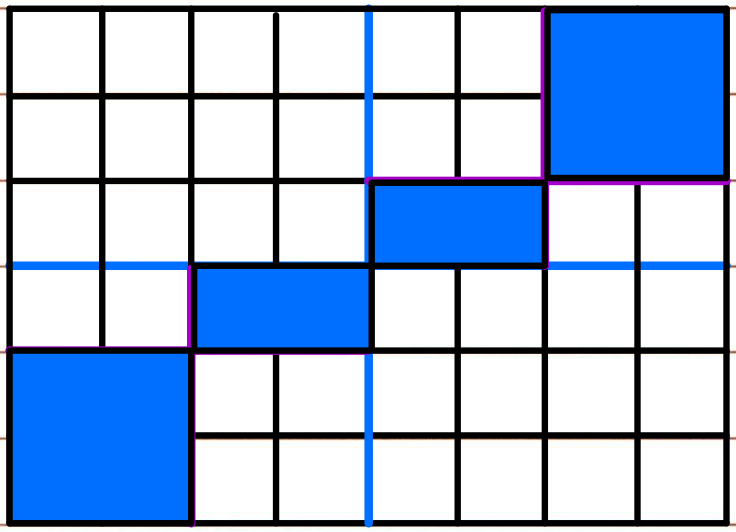
takes cmn operations

No. of operations at the second level



takes cmn/2 operations

No. of operations at the third level



takes $cmn/4$ operations

Total no. of operations $=$

$$
\begin{aligned}
& cmn + \\
& \tfrac{1}{2}cmn + \\
& \tfrac{1}{4}cmn + \\
& \vdots \\
\hline
& 2cmn \\
= \ & \Theta(mn)
\end{aligned}
$$

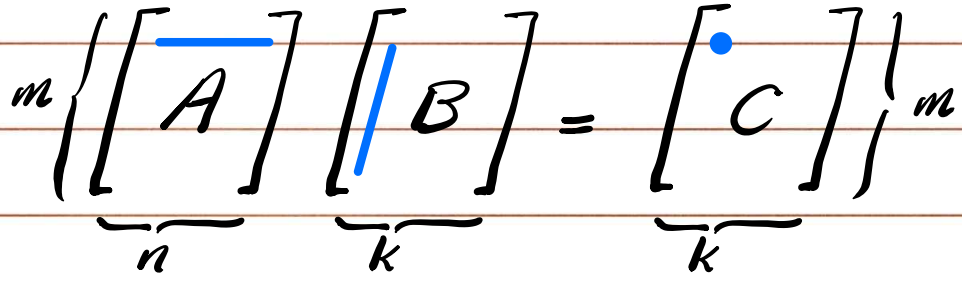
Matrix Chain Multiplication

Problem Statement

Given a sequence of n dense matrices along with their dimensions, find the order of matrix multiplications that minimizes the total number of element multiplications.

$$A = A_1 \cdot A_2 \cdot A_3 \ \ldots \ A_n$$

Recall the no. of element multiplications for two rectangular dense matrices



Requires mnk element multiplications.
(for the basic method_not using Strasson's alg.)

Ex.:

B.C.D

- B is 2x10
- C is 10x50
- D is 50x20

B.(C.D) results in 10,400 element multiplications.

(B.C).D results in 3,000 element multiplications.

Consider the sequence of matrices A_i thru A_j

$$(A_i \ \ldots \ A_k) \cdot (A_{k+1} \ \ldots \ A_j)$$

Observation:
The last multiplication in this sequence must involve the multiplication of the results of $(A_i \ \ldots \ A_k)$ and $(A_{k+1} \ \ldots \ A_j)$ where $i \leq k < j$

Let $OPT(i,j)$ = The optimal cost of multiplying matrices $i \ldots j$

Recurrence Formula 2:

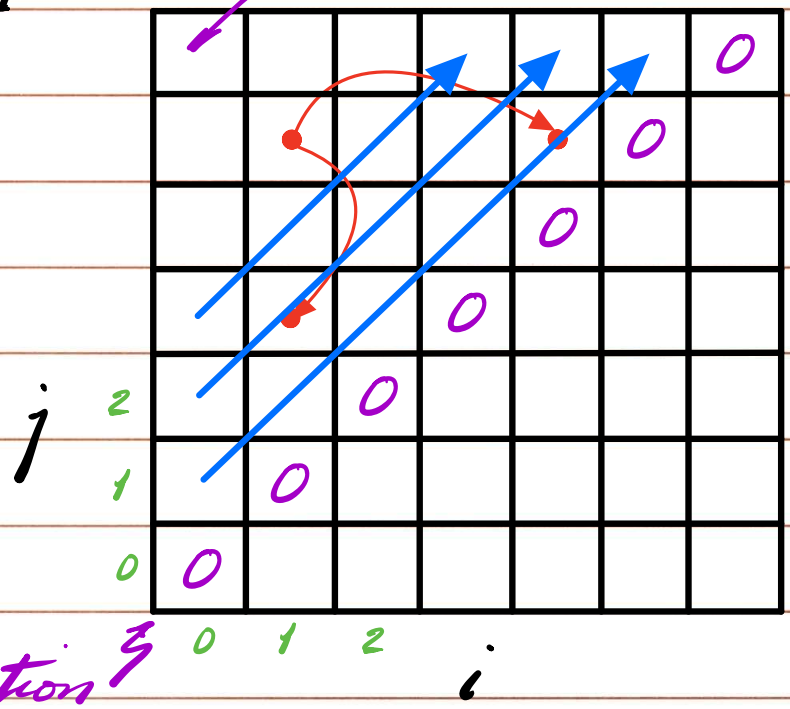
$$OPT(i,j) = \min_{i \leq k < j} \{ OPT(i,k) + OPT(k+1,j) \} + R_i C_k C_j$$

where R_i = No. of rows of A_i,
C_k = No. of columns of A_k
C_j = No. of columns of A_j

Bottom up pass:

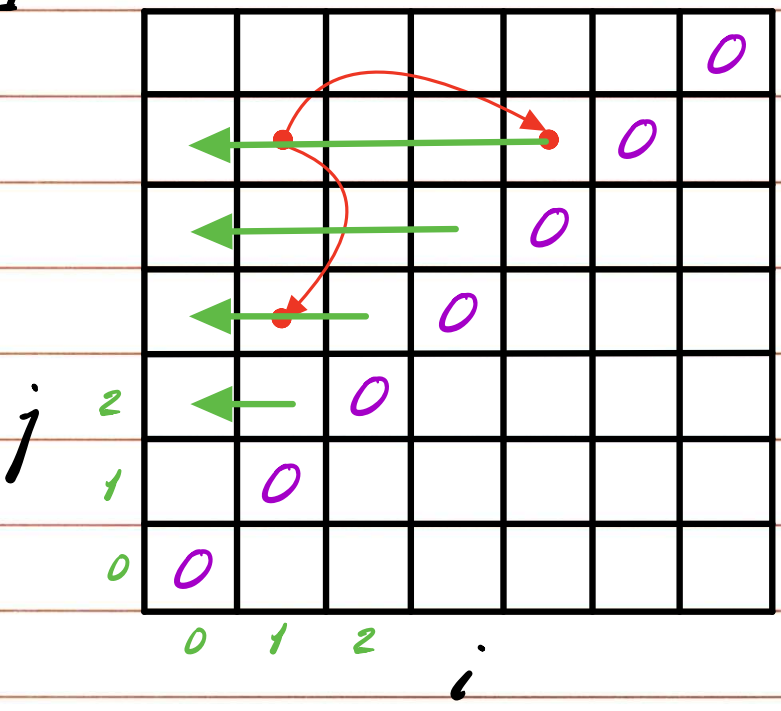
Value of the opt. solution

Option I

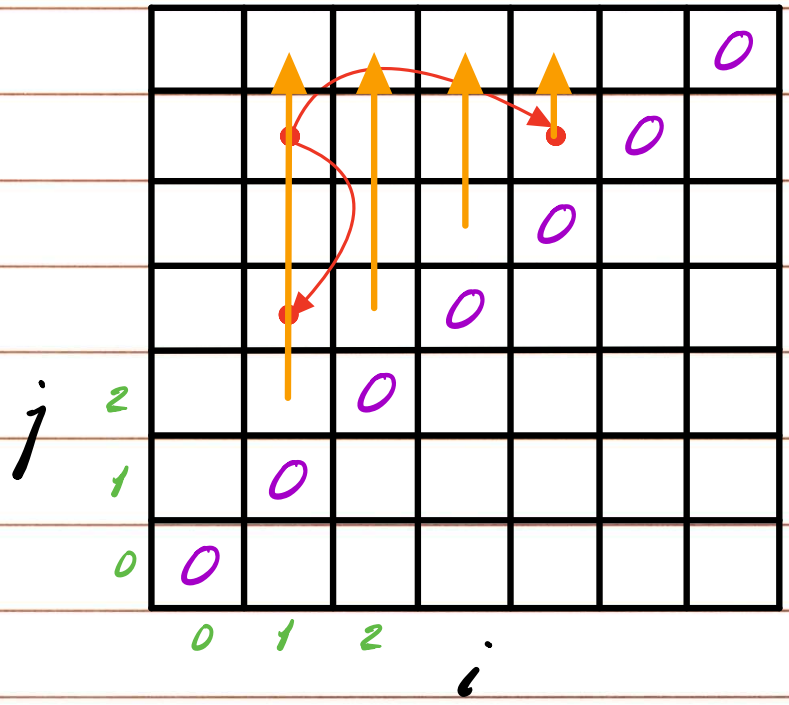


Initialization

Option II



Option III



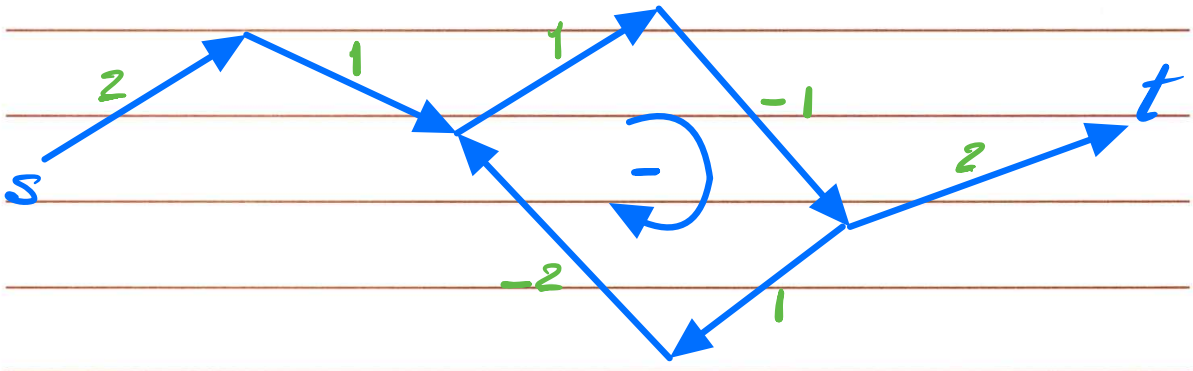
Pseudocode for Option II:

```
for i=1 to n, OPT(i,i) = 0
for j=2 to n
    for i=j-1 to 1 by -1
        Use recurrence formula 2
    endfor
endfor
```

Runs in $O(n^3)$ time

Is this an efficient solution? Yes!

Shortest Path Problem
(Dynamic Programming)



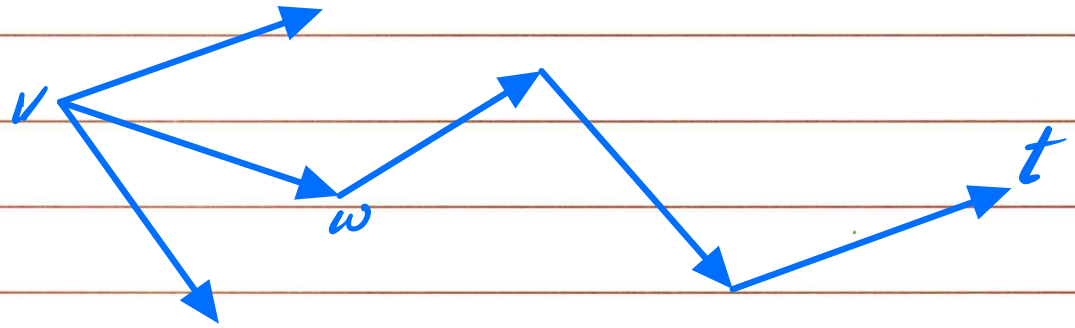
What is the shortest distance from s to T?

$-\infty$

If G has no negative cycles, then there is a shortest path from s to t that is simple and hence has at most n-1 edges.

Let OPT(i,v) denote the minimum cost of a v-t path using at most i edges

Objective : To find OPT(n-1, s)

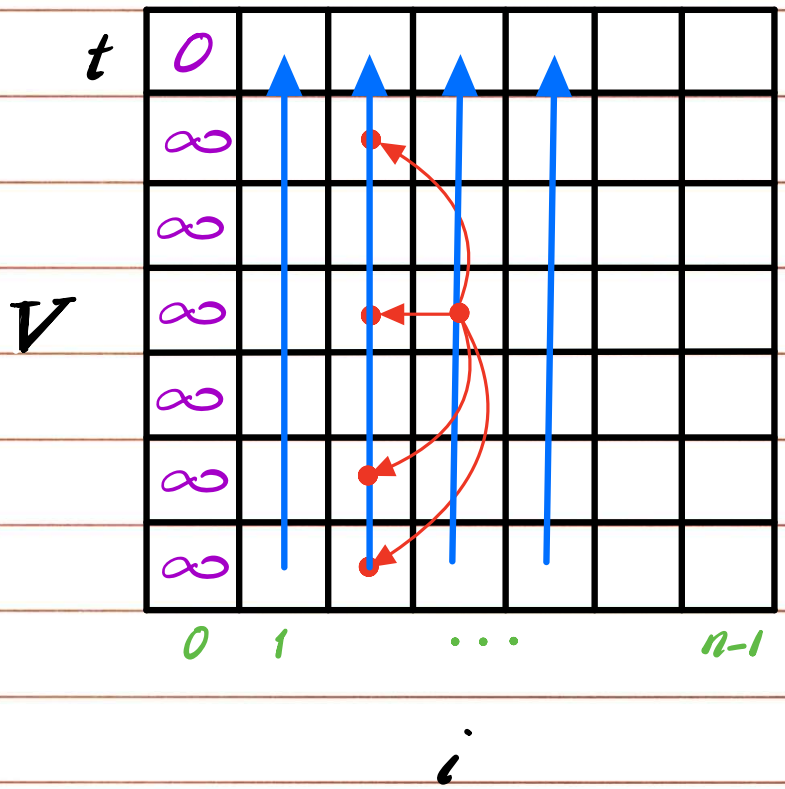


$$OPT(i,v) = \min_{w \in Adj(v)} \left(OPT(i-1,w) + c_{vw} \right)$$

But since the shortest path could have less than i edges, then

$$OPT(i,v) = \min\left(OPT(i-1,v),\ \min_{w \in Adj(v)}\left(OPT(i-1,w) + c_{vw}\right)\right)$$

(Recurrence Formula 3)



Bellman-Ford Alg.

Shortest Path (G, s, t)

n = no. of nodes in G

Define $OPT(0,t) = 0$, $OPT(0,v) = \infty$

for i = 1 to n-1 — $O(n)$

  for $v \in V$ in any order — $O(n)$

    Use recurrence formula 3 — $O(n)$

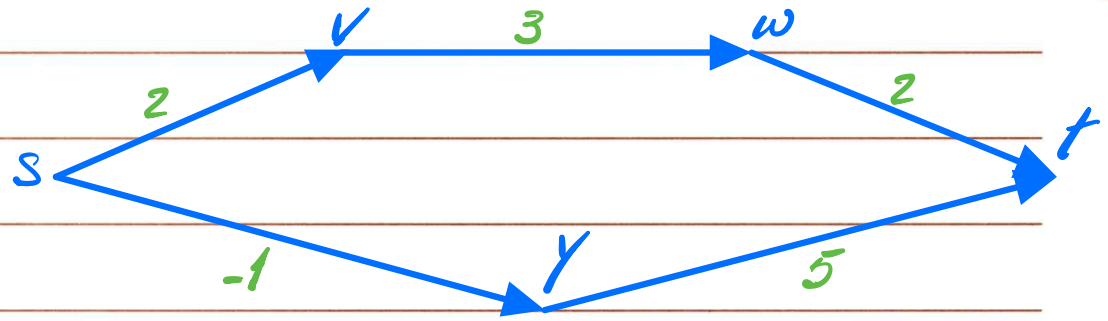
  endfor

endfor

Worst-case run time: $O(n^3)$

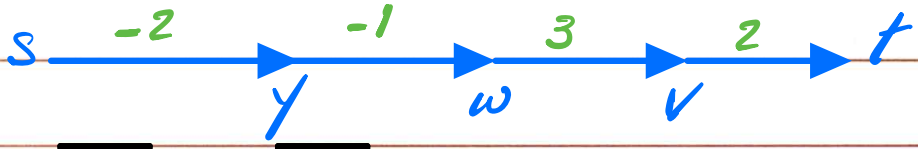
Tight bound is $\Theta(mn)$, since the total no. of comparisons in the inner loop cannot exceed m.

Ex.



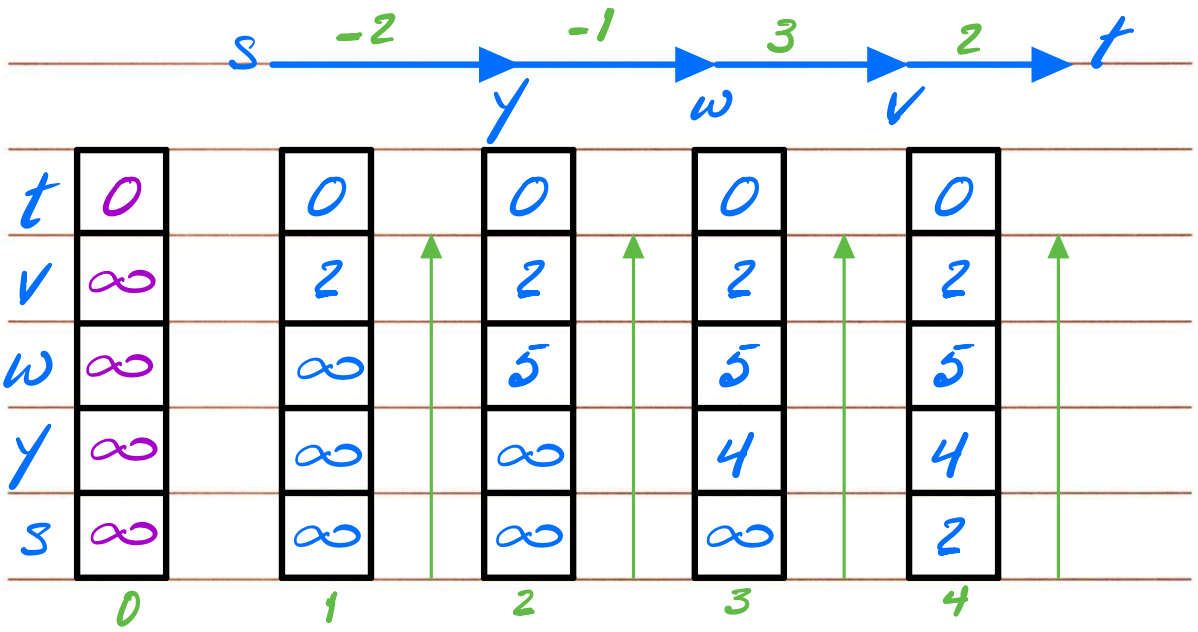
	0	1	2	3	4
t	0	0	0	0	
y	∞	5	5	5	
v	∞	∞	5	5	
w	∞	2	2	2	
s	∞	∞	4	4	

what happens if we only use one column?



	0	1	2
t	0	0	0
v	∞	2	2
w	∞	5	5
y	∞	4	4
s	∞	2	2

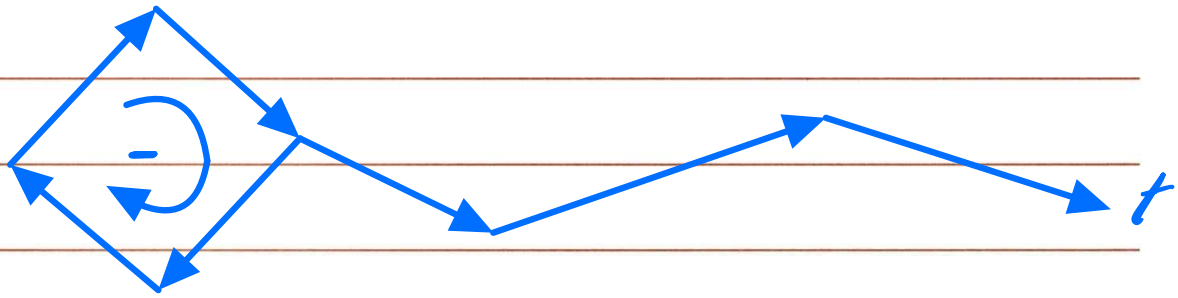
updating values in this order (from t to s), we are done in 2 iterations



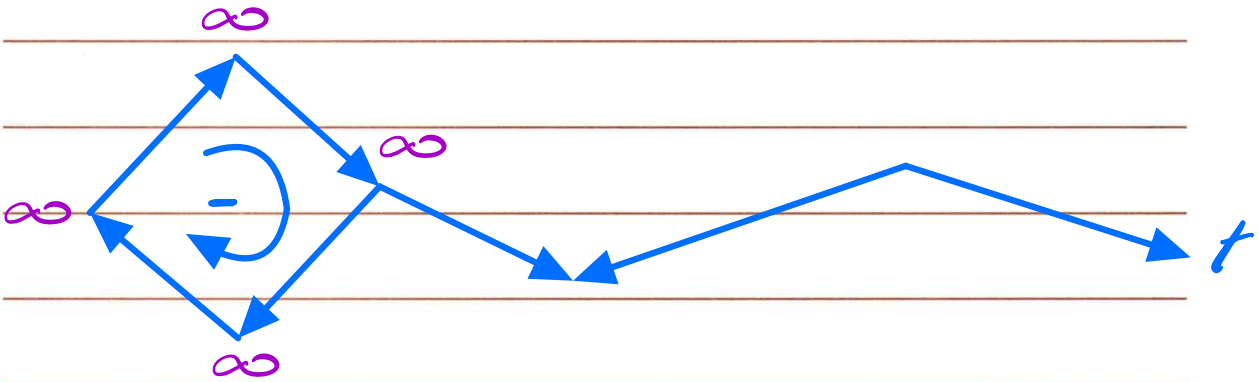
updating values in this order (from s to t), we will be done in 5 iterations. (exactly n-1 iterations)

So, to allow for faster propagation of information, we need to update/visits nodes in increasing order of their distance (in terms of no. of edges) to t.

How can we detect if a directed graph has a negative cycle?

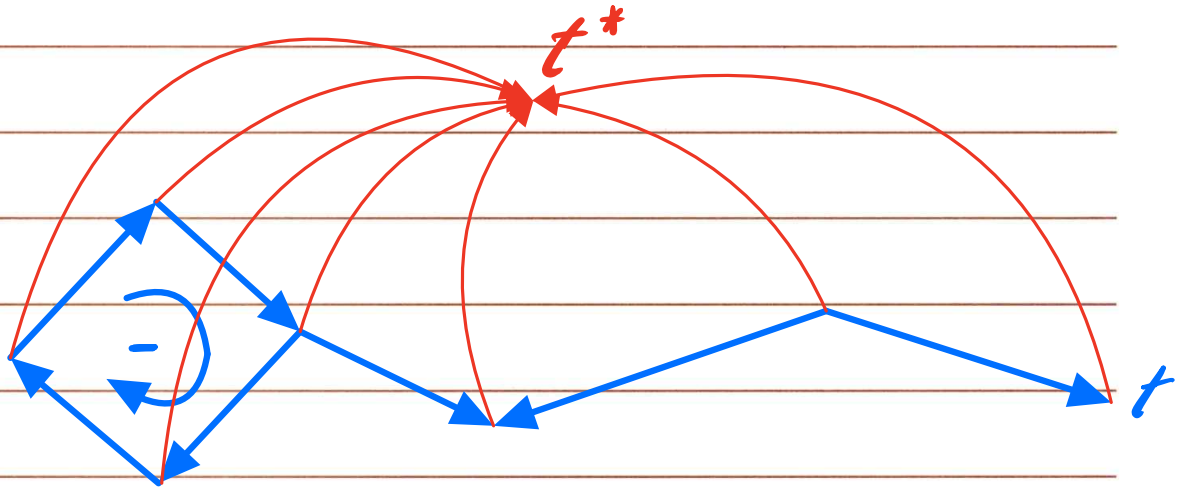


If there is a path from the negative cycle to t, distances to t for nodes on that cycle will keep decreasing even after n-1 iterations.



But if there is no such path to t, the initial distances of ∞ never go down after n-1 iterations

To ensure that any negative cycle is detected, we will create a new destination node t^* and connect all nodes to it.



Bellman-Ford	Dijkstra
$O(mn)$	$O(m \lg n)$

Do we always choose Dijkstra's alg. when edge weights are positive?

Bellman-Ford is much more suitable for distributed parallel processing.