PAGE No.
DATE / /

AOA week 8 Notes

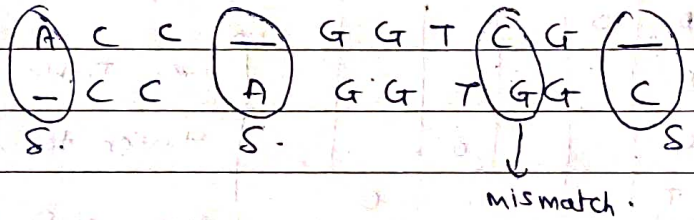
(*) Sequence Alignment Problem.

DNA Strand consists of 4 Bases {A, C, G, T}

Comparing DNA Strands

S_1 = A C C G G T C G

S_2 = C C A G G T G G C

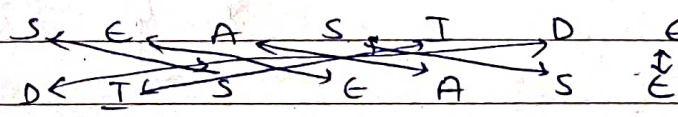


3 gaps 1 mismatch.

$X = \{x_1, x_2 \ldots x_m\}$

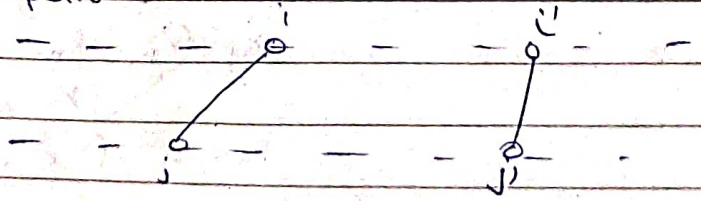
$Y = \{y_1, y_2 \ldots y_n\}$

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



perfect matching But they are not similar.

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



(i, j) $(i', j') \in M$

if $i < i' \Rightarrow j < j'$

For an alignment M between X and Y.

1. we incur a "gap penalty" of δ For each gap
   ↳ Fixed

2. For each mismatch (of letters $p \neq q$) we incur a mismatch cost α_{pq}
   ↳ not Fixed.

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

α Table given to us.

$\alpha_{xx} = 0$

$\alpha_{xy} \Rightarrow$ +ve int.

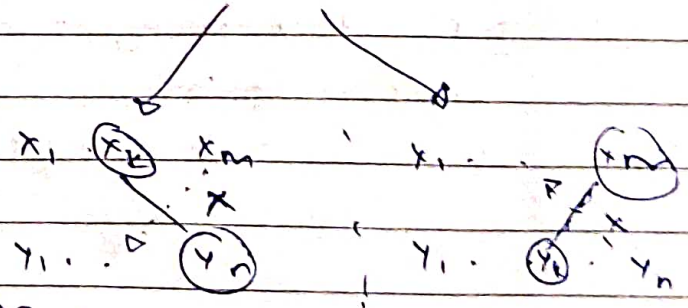
Defn Similarity between strings X and Y is the minimum cost of an aligament between X and Y.

$$X = \{ x_1 \ldots x_m \}$$

$$Y = \{ y_1 \ldots y_n \}$$

Say M is OPT soln.

$(x_m, y_n) \in M$ or $(x_m, y_n) \notin M$



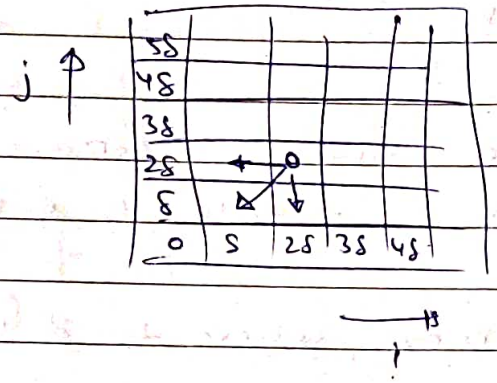
if (x_k, y_n) exist so x_m will not be paired as it will result in cross pair

if (x_m, y_k) exist so y_n will not be paired as it will result in cross pair.

PAGE NO.
DATE / /

Rec Formula ①

$$OPT(m,n) = \min \begin{cases} OPT(m-1, n-1) + \alpha_{x_m y_n} \\ OPT(m-1, n) + \delta \\ OPT(m, n-1) + \delta \end{cases}$$



Alignment (X, Y)

Init $A[i, 0] = i\delta$ for each i

Init $A[0, j] = j\delta$ for each j

For j=1 to n

For i=1 to m

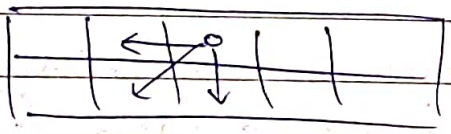
∴ Rec Formula ①

Time : $O(m * n)$

Space : $O(m * n)$

Suppose DNA seq ≈ 3B size

Space Req : 3B * 3B



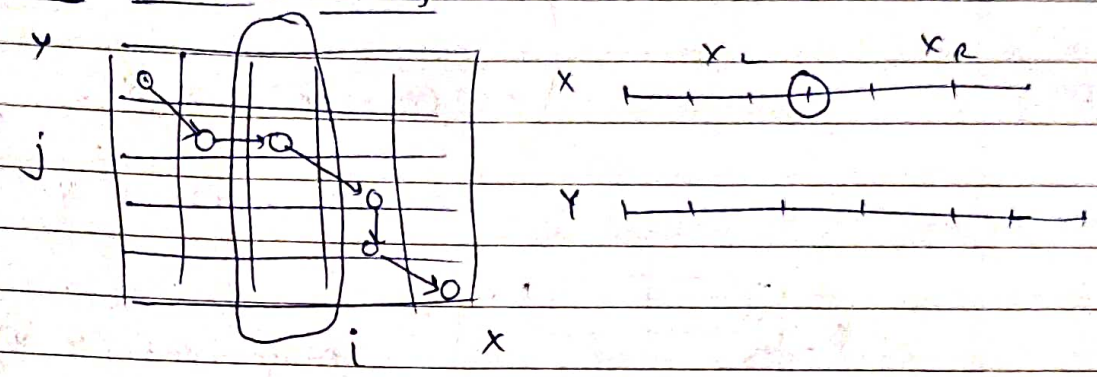
But for calculation of a particular row we only need 1 prev row.

But if we keep 2 rows to solve.

We will get the optimal ans but how will we find opt SOLN as we will not be able to backtrack.

PAGE No.
DATE / /

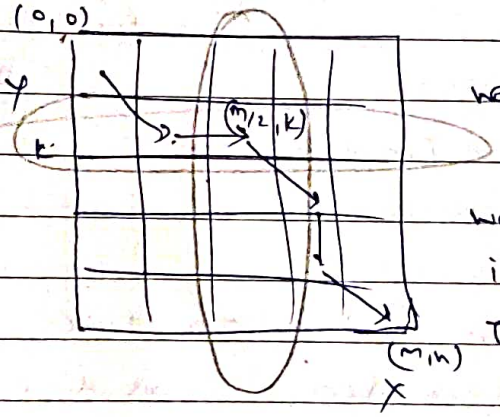
Memory Efficient Way.



We will use Divide & Conquer

We will divide X in middle to get X_L and X_R

But how do we know where to divide Y



We have x in middle.

We now need to find index 'k' such that the optimal soln passes through that point

∴ OPT path passes through index k

where $x = m/2$

i.e $\left(\frac{m}{2}, k\right)$

let $f(q,k)$ $\quad$ $g(q,k)$

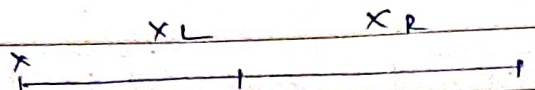
↓ $\quad$ ↓

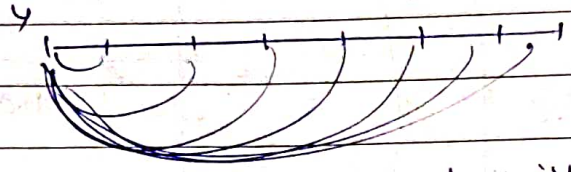
OPT $((0,0) \rightarrow (q,k))$ $\quad$ OPT $((q,k) \rightarrow (m,n))$

∴ k is the index which minimizes $\left(f(q,k) + g(q,k)\right)$

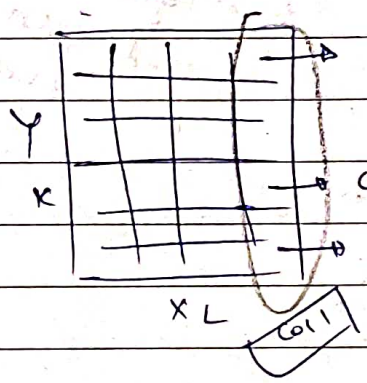
PAGE No.
DATE / /

eg :





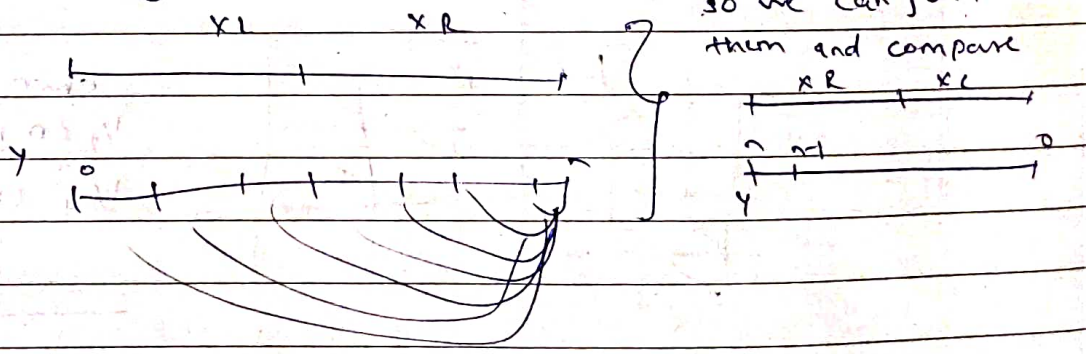
- x_L could be solved with y_0
  x_L " y_1
  x_L " y_n



Cost of alignment for x_L and $y_{0..n}$

Cost of alignment for x_L and $y_{1..k}$

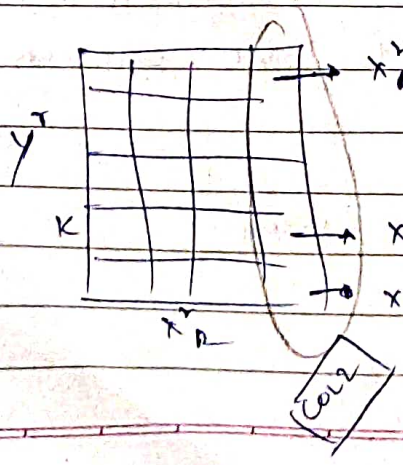
Cost of alignment for x_L and y_0



x_R could be aligned with y_n

" $y_{n-1..n}$

" $y_{k..n}$.



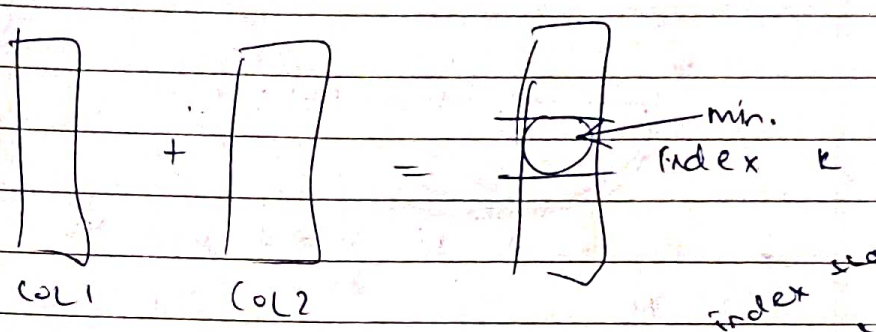
x^r_L and all of y.

x^r_L and $y_{n-k}..y_n$

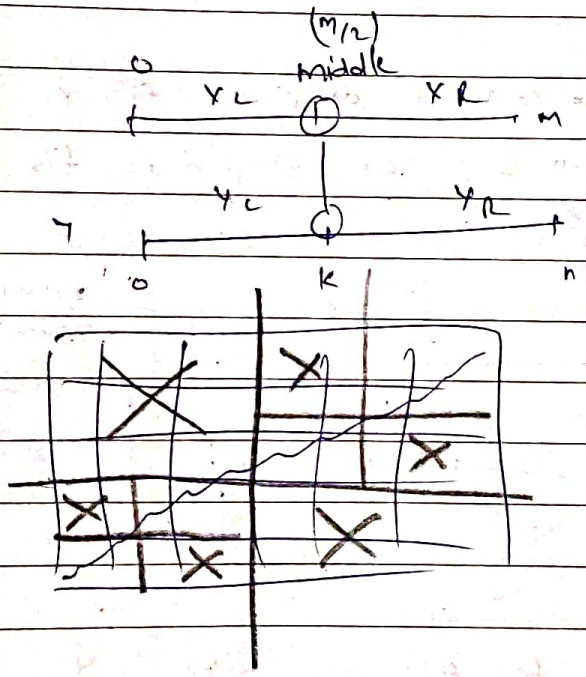
x^r_L and none of y

PAGE No.
DATE / /

after getting COL1 and COL2



Find the index that give min ans and split 4 on that index



cmn
1/2 cmn
1/4 cmn
.
.
.

2 cmn

Time : O(2cmn)

As we are just interested in final 2 columns of X_L Y and X_R Y we use memory efficient way

Space O(m+n) linear.

memory efficient way

Divide ← use Dynamic programming to find optimal split point for y

Solve

Conquer ← just concatenate the ans of subproblems.

	memory wasteful way	memory efficient way
Time	$\Theta(mn)$	$O(2mn)$
Space	$O(mn)$	$O(m+n)$

In D&C we solve for OPT SOLN so we directly get OPT SOLN after concat

PAGE No
DATE / /

⑧ matrix chain Multiplication

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

Eg

$B \cdot C \cdot D$

$B = 2 * 10$

$C = 10 * 50$

$D = 50 * 20$

$B \cdot C \cdot D = (B \cdot C) \cdot D \Rightarrow 3000$ operations

or

$B \cdot (C \cdot D) \Rightarrow 10400$ operations.

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

Finally

$$((1 \text{ matrix})(1 \text{ matrix}))$$

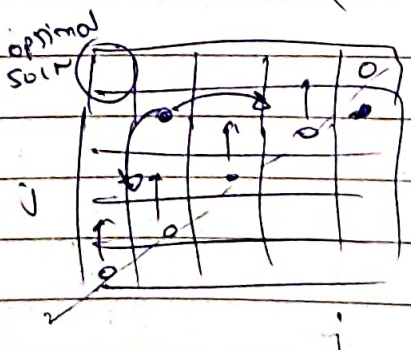
we need to find where to split

$OPT(i,j)$ = optimal cost of multiplying matrices i through j

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

Formula

$(R_i C_k C_j)$: final product cost



PAGE NO.
DATE / /

For i=1 to n OPT(i, i) = 0

~~For j=t to~~

For j=2 to n

  For i = j-1 to 1 dec by -1

    Rec formula (2) → O(n)

  end for

end for

O(n) × O(n) × O(n)

$O(n^3)$ Time

Efficient Soln.

(*) Shortest path problem Dynamic Programming.

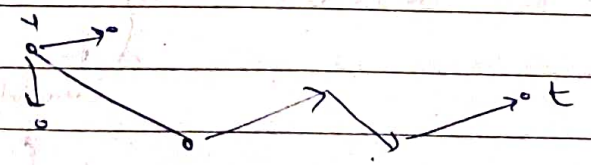
Assumption: No -ve cost cycles

There can be -ve cost edges.

If G has no -ve cycles, then there is a shortest path from s to t that is simple and hence atmost 'n-1' edges.

OPT(i, v) ⇒ min cost from v-t with atmost i edges

Problem statement: OPT(n-1, s)

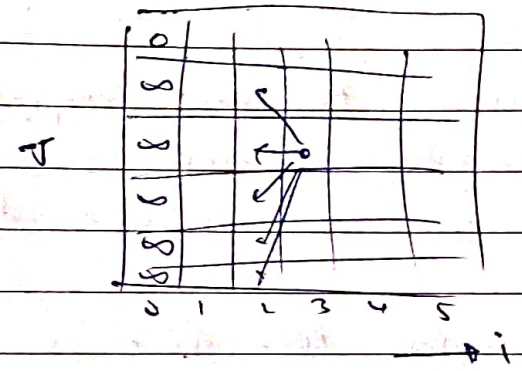


$$OPT(i, v) = \min_{\text{For } w \text{ in adj}(v)} \{ (v_w + OPT(i-1, w)) \}$$

PAGE NO.
DATE / /

But by the recurrence relation we are only considering paths of exactly (or 'i')
Path are 4 to 6 atmost 'i'
So they can also be less than 'i'

$$OPT(i,v) = \min \begin{pmatrix} OPT(i-1,v) \\ \min_{\substack{\text{for} \\ w \in Adj(v)}} \{ c_{vw} + OPT(i-1,w) \} \end{pmatrix}$$



Bellman Ford Algo.

```
Shortest-path (G, s, t)
    n = no. of nodes in G
    define M[0, t] = 0      M[0, v] = ∞
    For i = 1 to n-1
        For v ∈ V in any order
            M[i, v] = min( M[i-1, v],
                           min_{w ∈ Adj(v)} { c_vw + OPT(i-1, w) } )
        End For
    End for
```

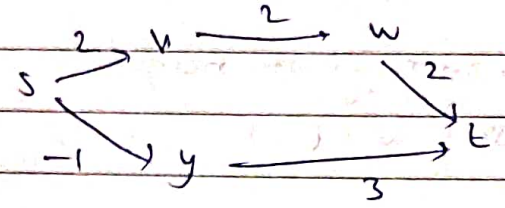
O(n) — For v ∈ V in any order
O(n) — For i = 1 to n-1
O(n) — min_{w ∈ Adj(v)} { c_vw + OPT(i-1, w) }
O(mn) — overall

But every edge can be called only once (SAME AS THAT OF DIJKSTRA)

$O(m \cdot n)$

at max $m = n^2$

eg :



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

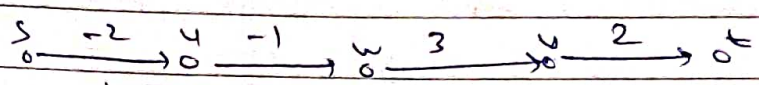
→ 2 col's same.

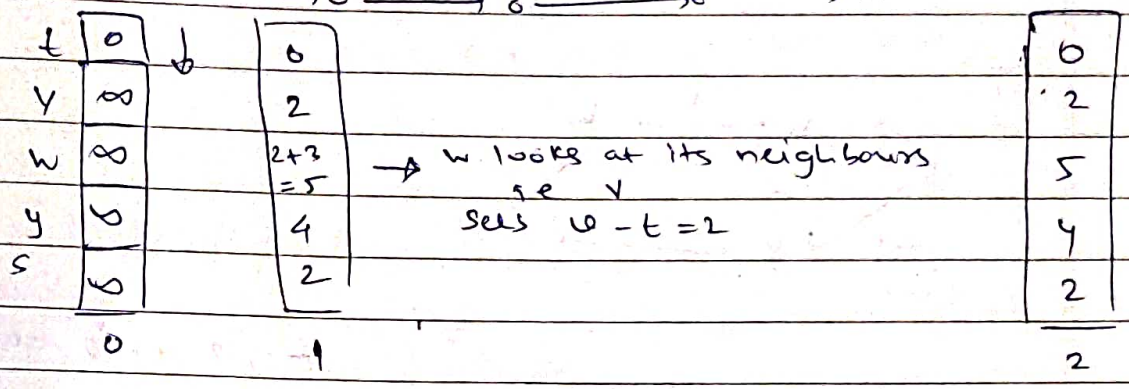
STOP HERE

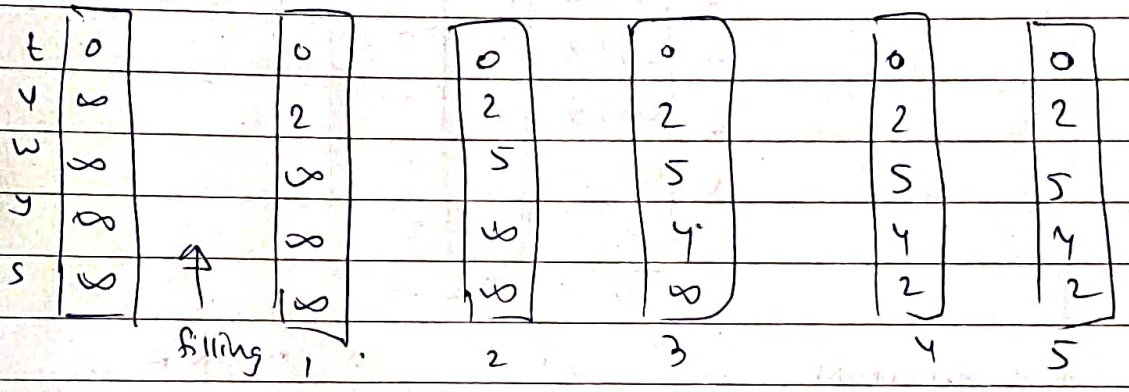
1 COLUMN (Search t with almost 1 edge)		2 COLUMN (search t with almost 2 edge)	
t-t	0	t-t	0
y-t	3	y-t	3
w-t	2	w-t	2
v-t	∞	v-t	4
s-t	∞	s-t	2

At any time only last 2 col's are in consideration i.e col3 can be judged from col2 only.

Trying to solve with only 1 column.







O(n-1) worst case

What is preference of filling?

BFS order from T

As its neighbours will convey info

then their neighbours

:

so on.

We found shortest dist

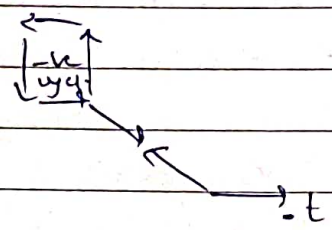
For shortest path we just keep a track of present

i.e neighbour that give shortest distance to t.

∴ shortest path always found in 'n-1' iterations

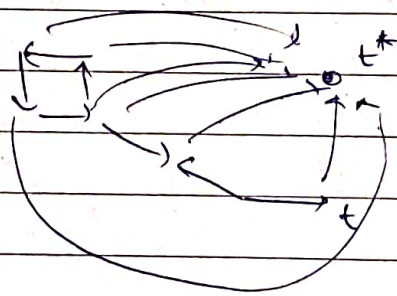
if after n-1 iterations value go down this indicates presence of -ve cycle.

Eg: How to FIND -VE CYCLE



In this running it after n-1 times next change column value as -ve cycle is not on path to t

To FIND CYCLE



~~But~~ Add new node t^* connect every node to t^* run Bellmanford. For t^*

Bellman Ford	Dijkstras
$O(mn)$	$O(m \log n)$

If -ve cost edges can only use Bellman Ford

If no -ve cost edges
Can use Bellmanford & Dijkstras.

Some cases Bellmand ford faster than Dijkstras as it can be easily parallelized.