Dynamic Programming

Sequence Alignment Problem

A DNA strand consists of a string of molecules called bases.

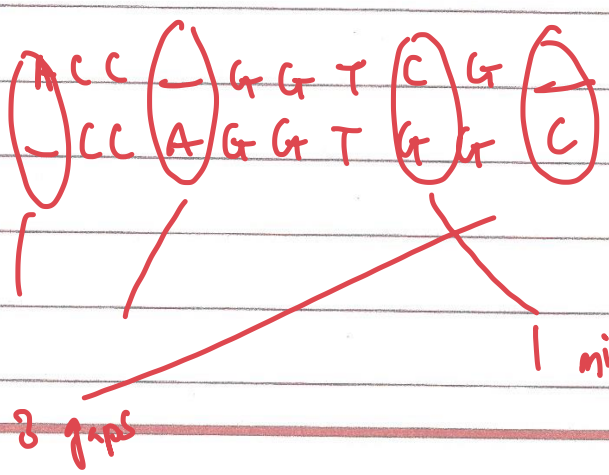
4 Types of bases:

- Adenine A
- Cytosine C
- Guanine G
- Thymine 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

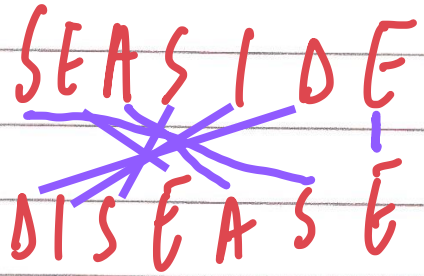


Suppose we have 2 strings X & Y.

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

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

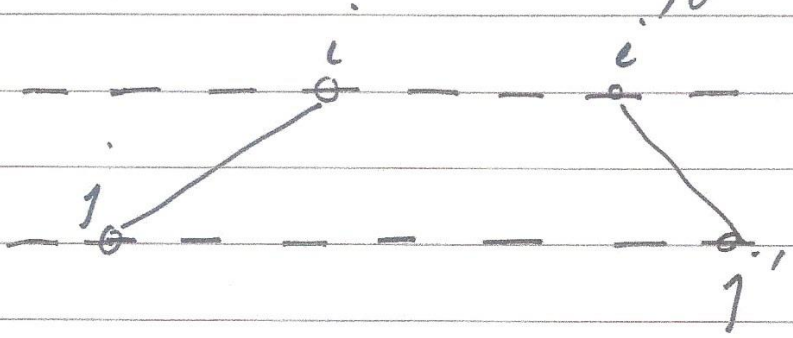
Def. A matching is a set of ordered pairs with property that each item occurs at most once.



perfect matching but they are not similar

Matching is an alignment with no crossing pairs.

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



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

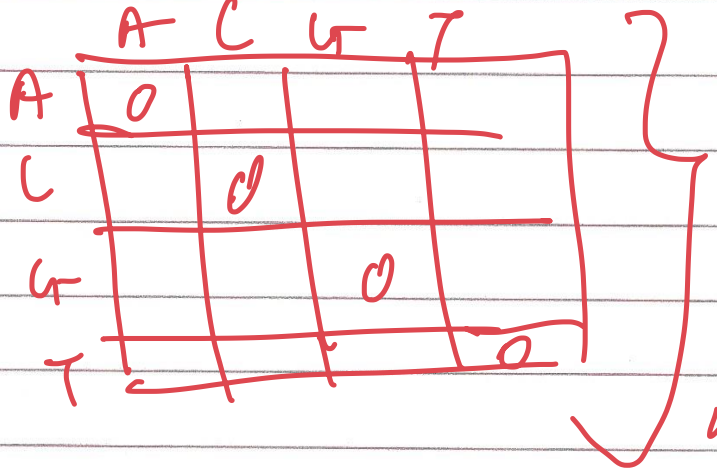
& $i < i' \Rightarrow j < j'$

For an alignment M between X & Y

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

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

Not Fixed; this penalty depends on which two characters mismatch



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

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

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

? Is this a pair in optimal solution or not

Say M is an opt. solution.

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

~~Define OPT(~~

$x_1 \ldots x_m$

$y_1 \ldots y_j \ldots y_n$

If (x_m, y_n) is not a pair

Consider x_m to be paired with some y_j

if this is case y_n cannot be paired as it would lead to cross pairing

Similarly is case if y paired then x not considered

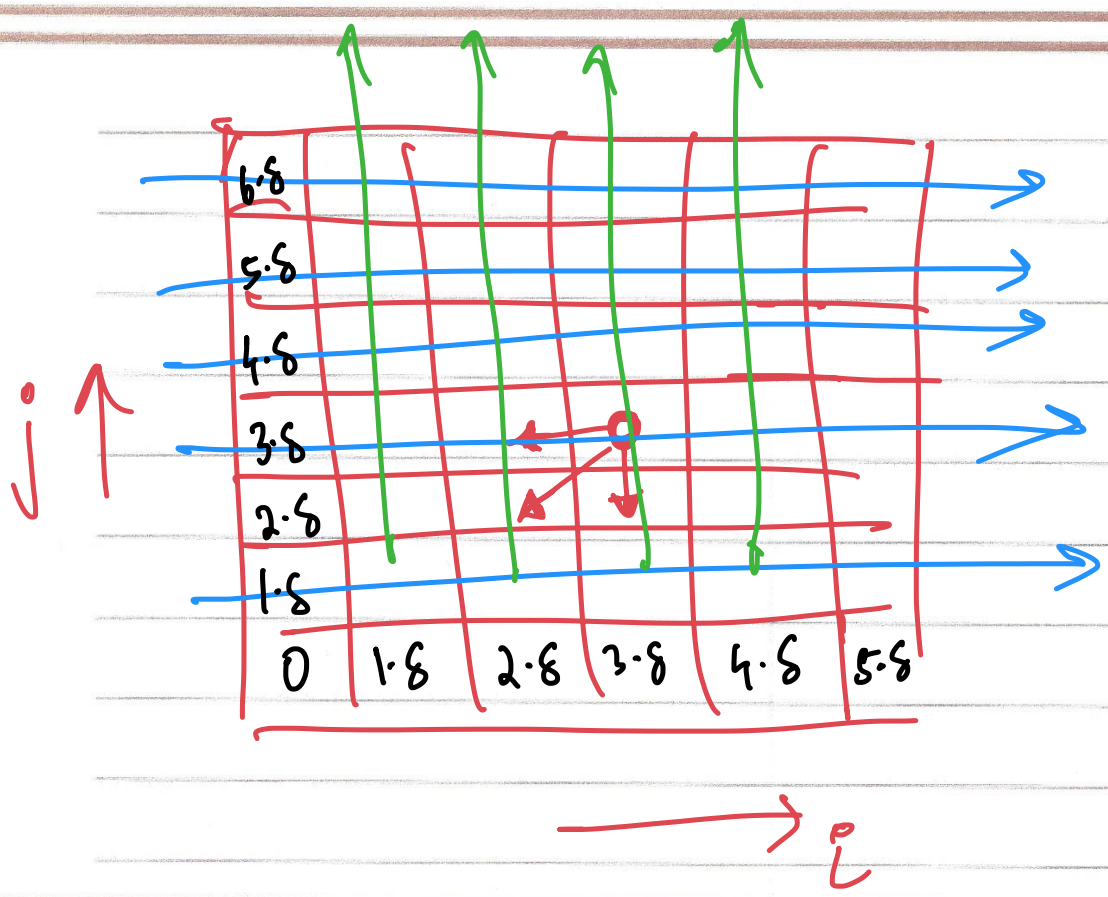
In an optimal alignment M, at least one of the following is true:

Recurrence formula ①

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

2) - x_m is not matched $\Rightarrow OPT(m,n) = OPT(m-1, n) + \delta$

3) - y_n is not matched $\Rightarrow OPT(m,n) = OPT(m, n-1) + \delta$



Alignment (X, Y)

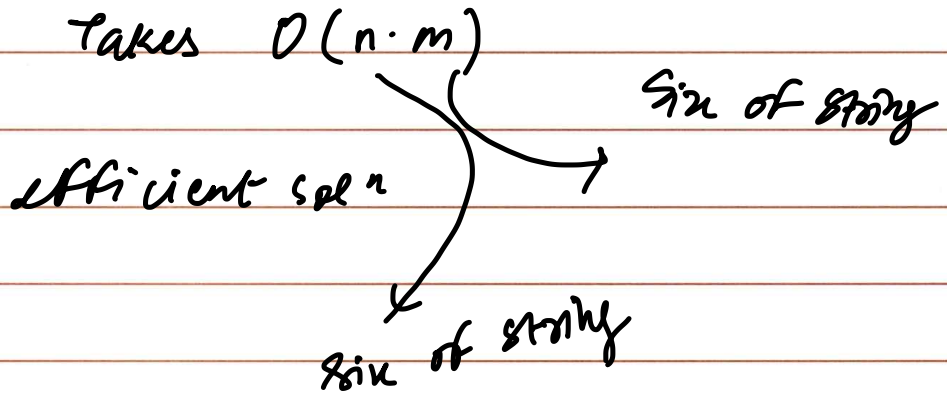
Initialize $A[i,0] = i\delta$ for each i

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

For $j = 1$ to n

For $i = 1$ to m

Recurrence Formula ①



Both represent size of some input

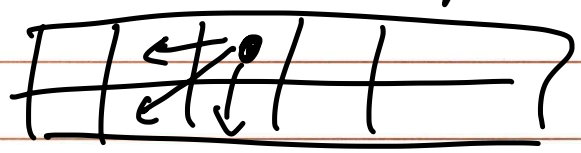
DNA sequence ≈ 3 billion size

∴ matrix size $(3 \text{ billion})^2$

MEMORY WASTEFUL WAY

But for calculation we only need 1 previous row

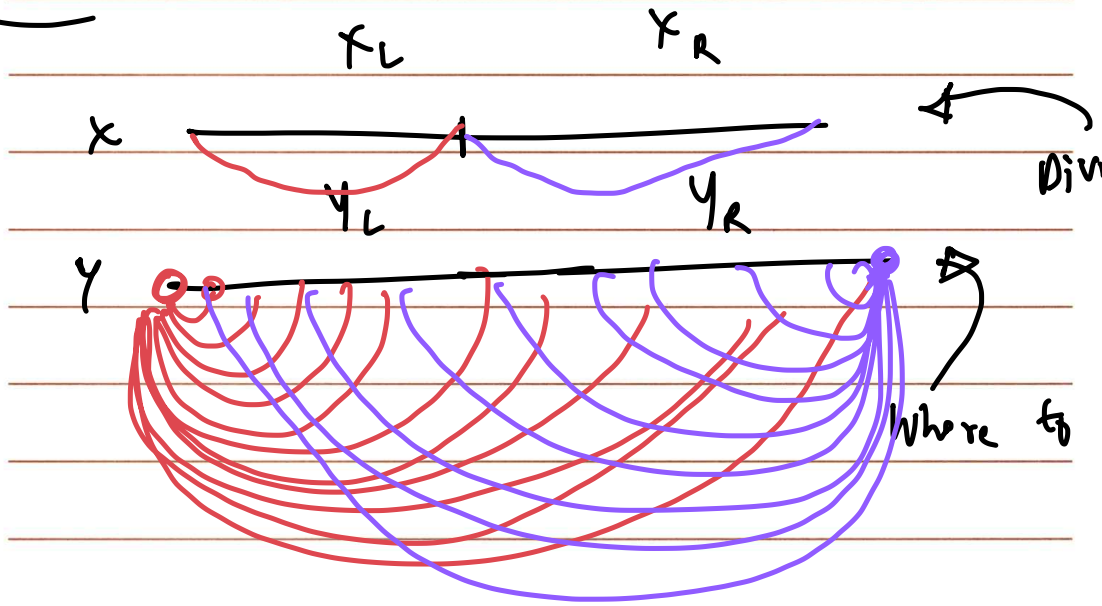
MEMORY EFFICIENT WAY

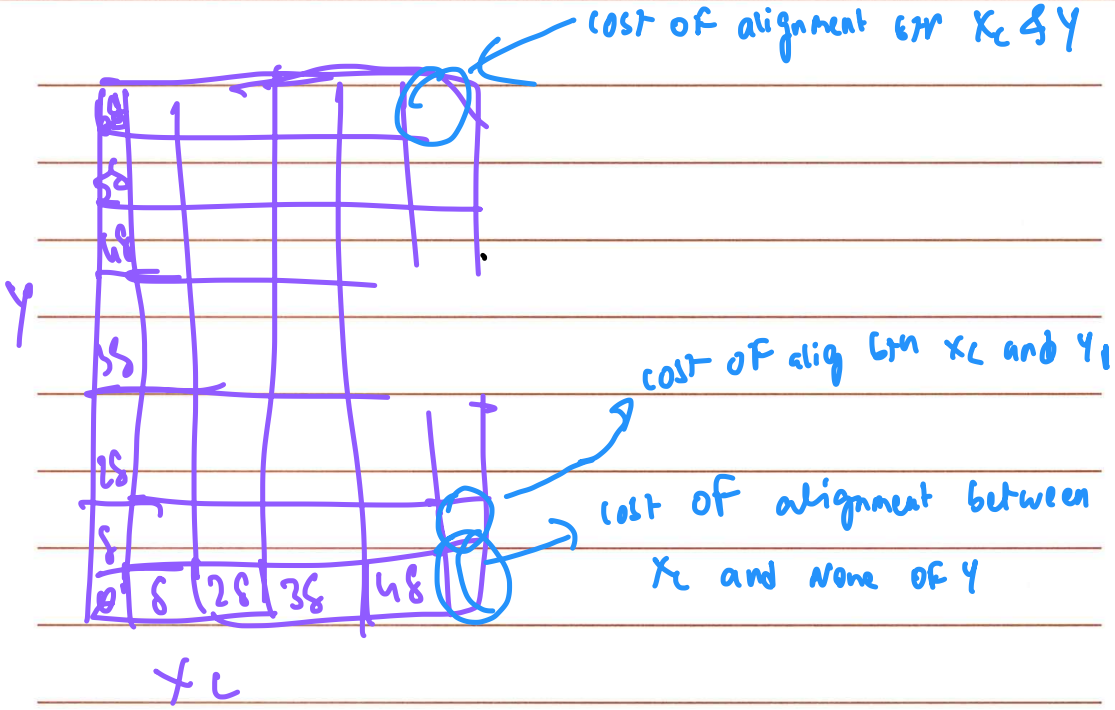


But if we only keep two rows, we won't be able to backtrack the answer (alignment)

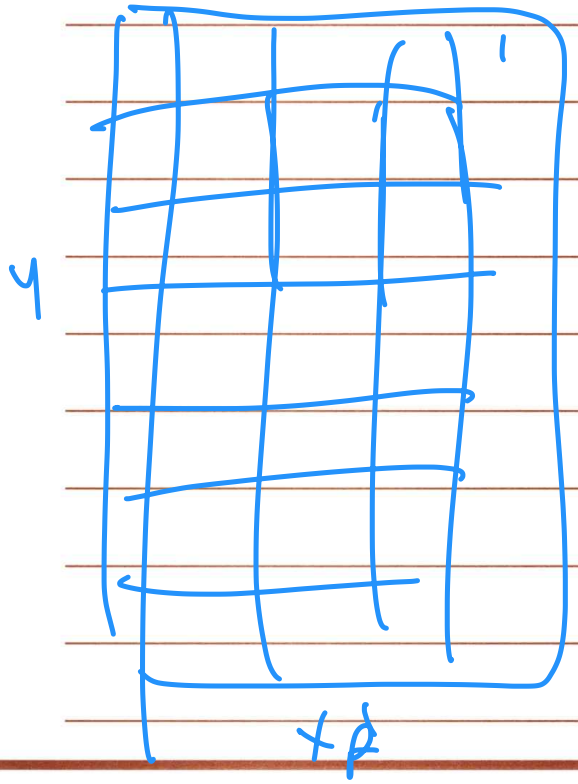
TRY TO FIND ALIGNMENT WITH MEMORY EFFICIENT WAY

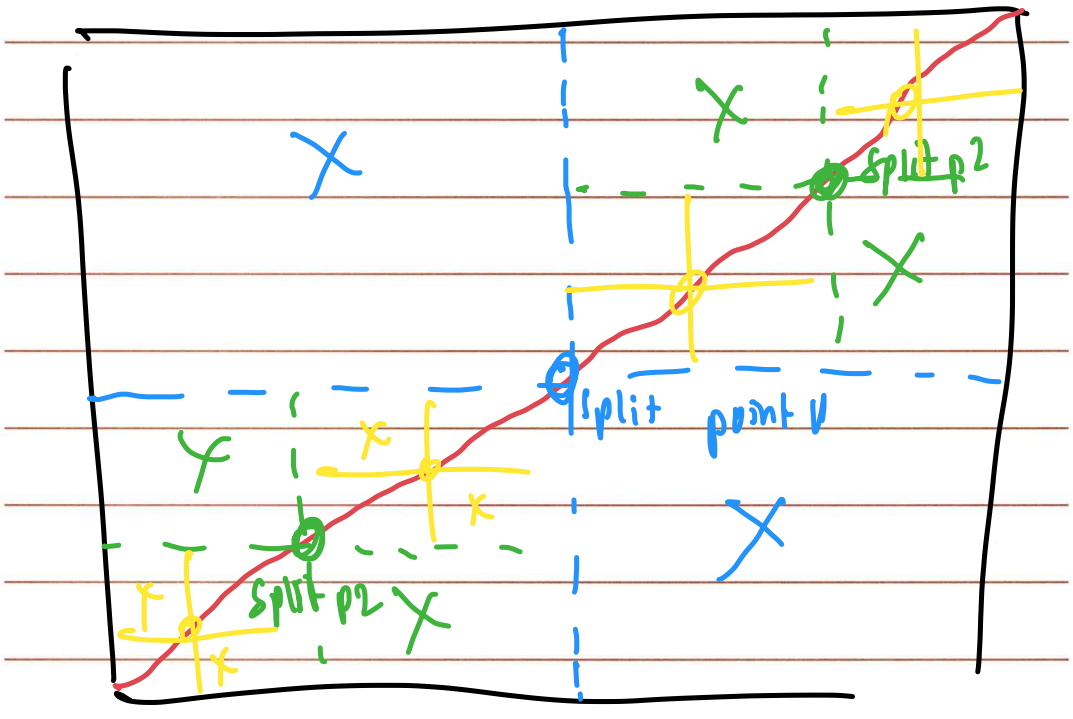
SOLUTION


x_L
x_R
x
y_L
y_R
y
Divide in middle
Where to divide y


cost of alignment btw x_L & y
y
cost of alig btw x_L and y_R
cost of alignment between x_L and none of y
8
28
38
48
x_L

op is at divide step





Cmn

$Cmn/2$

$Cmn/4$

$\vdots$

$2\,Cmn$

memory wasteful

Space: $3B + 3B$

Time: Cmn

memory Efficient

Space: $3B + 2$

Time: $2Cmn$

Matrix Chain Multiplcatio

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

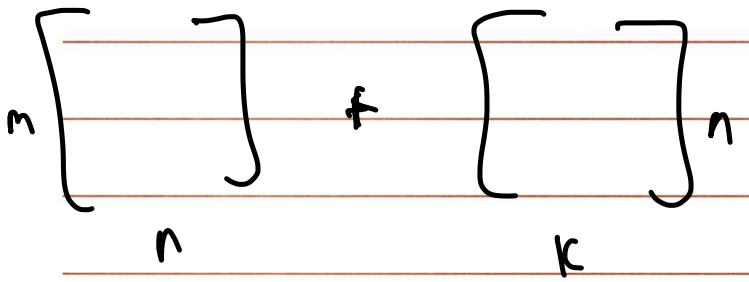
eg:

B · C D

B = 2 * 10

C = 10 * 50

D = 50 * 20



* operations = m * n * K

∴ $B \cdot C \cdot D = B \cdot (C \cdot D)$ ⇒ 10,400 operations

OR

$(B \cdot C) \cdot D$ ⇒ 3000 operations

$((A_i \cdot \ldots \cdot A_K)(A_{K+1} \cdot A_j))$

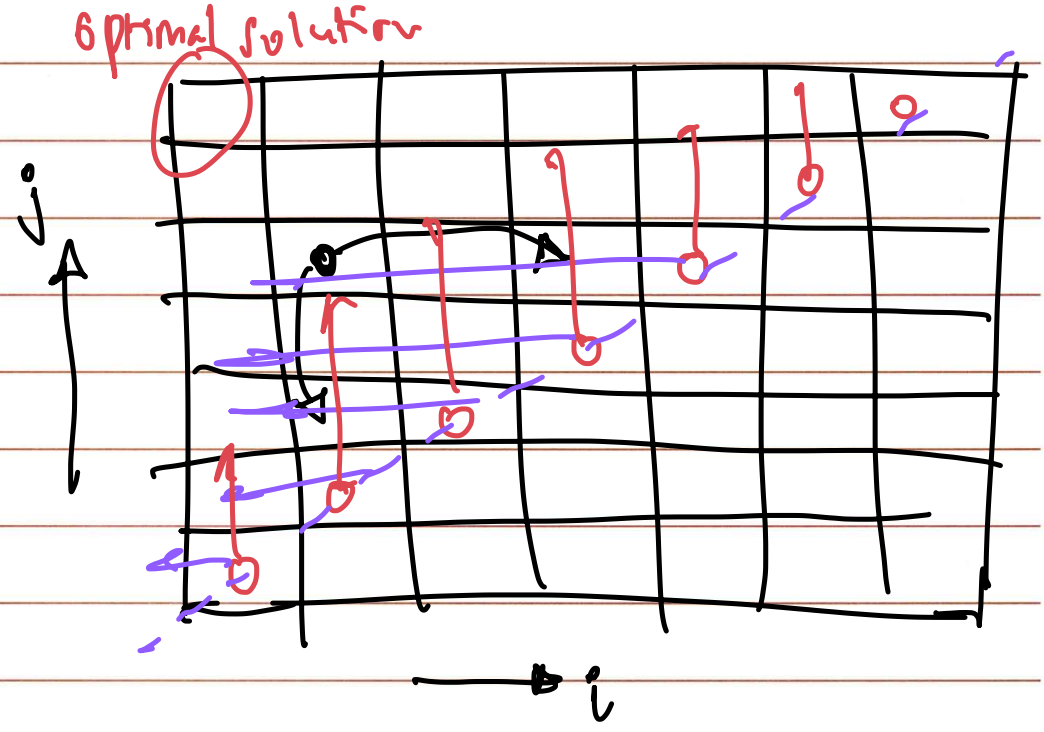
Final step $((\text{matrix} \quad \text{matrix}))$

We need to find where we need to split

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

$$OPT(i,j) = \min_{i \le k \le j} \left\{ OPT(i,k) + OPT(k+1,j) + \text{Final product cost } (R_i C_k C_j) \right\}$$

Rec Formula (2)



```
For i = 1 to n
    opt(i, i) = 0

For j = 2 to n                                          O(n)
    for i = j-1 to 1 by -1                              O(n)
        use rec formula (2)                             O(1)
    end for
end for
```

$O(n^3)$ Time

Efficient solution.

Shortest Path Problem

Dynamic Programming

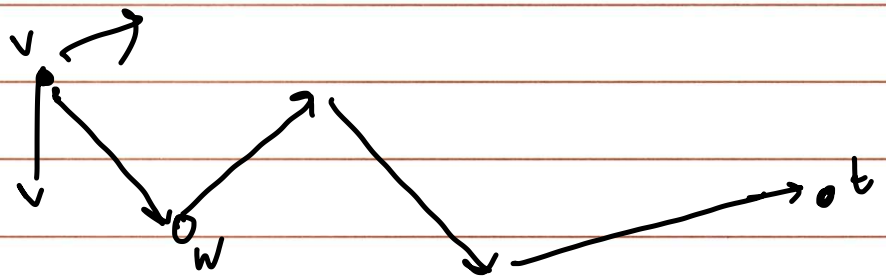
Assumption : No -ve cost cycles

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 at most n-1 edges

OPT(i, v) denotes the min cost of a v to t path using at most i edges

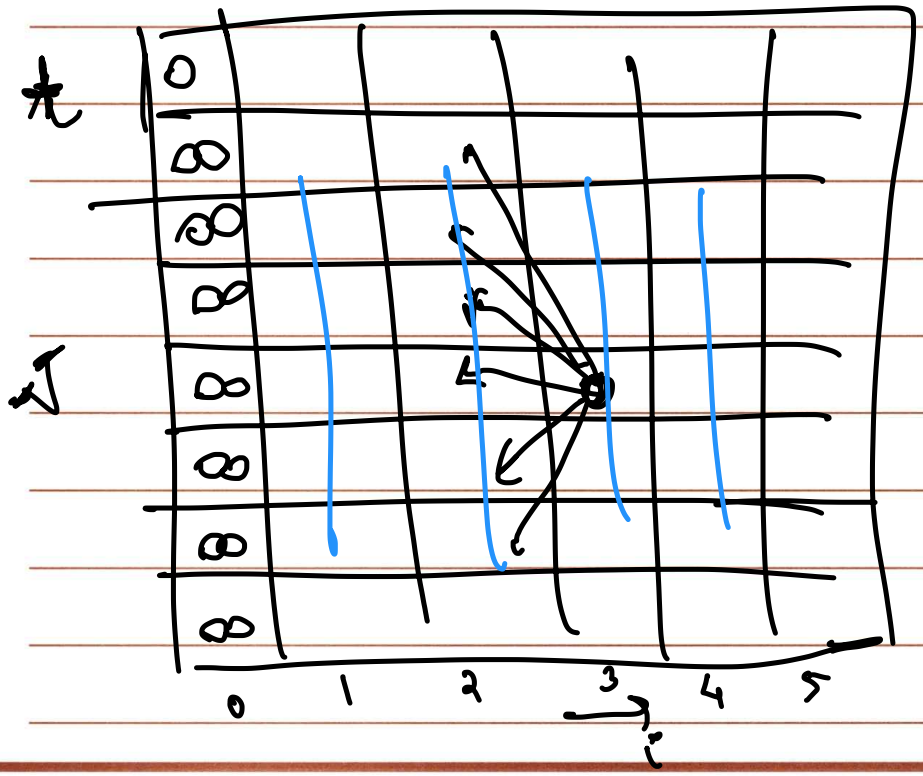
OPT(n-1, s)



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

This path should also have fewer edges than i

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



Apply recursion formula:

Bellman-Ford Alg.

Shortest-path (G, s, t)

n = no. of nodes in G

define $M[0,t] = 0$, $M[0,v] = \infty$

for $i = 1$ to $n-1$ — $O(n)$

  for $v \in V$ in any order — $O(n)$

    $M[i,v] = \min(M[i-1,v], \min_{(v,w) \in Adj(v)}(M[i-1,w] + c_{vw}))$ — $O(n)$

  end for

end for

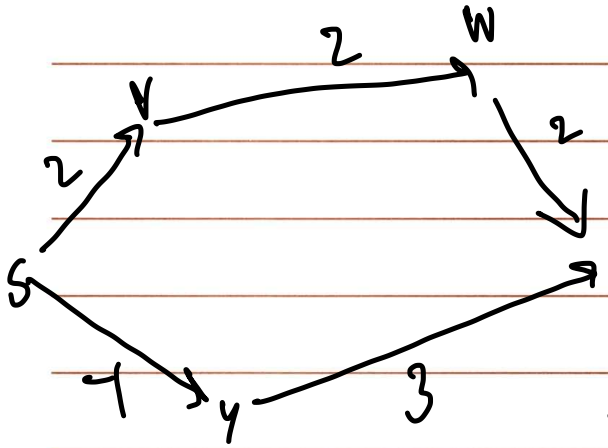
$O(n^3)$

But each edge can only be called once

(SAME AS THAT OF DIJKSTRA'S)

$O(n \cdot m)$

efficient solution



n-1 iterations

2 col's same

All to Bellman Ford Algo stop here

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

1 column (reach t with 1 edge)

t-t 0

y-t 3

w-t 2

v-t ∞

s-t ∞

2 column (reach t with 2 edge)

t-t 0

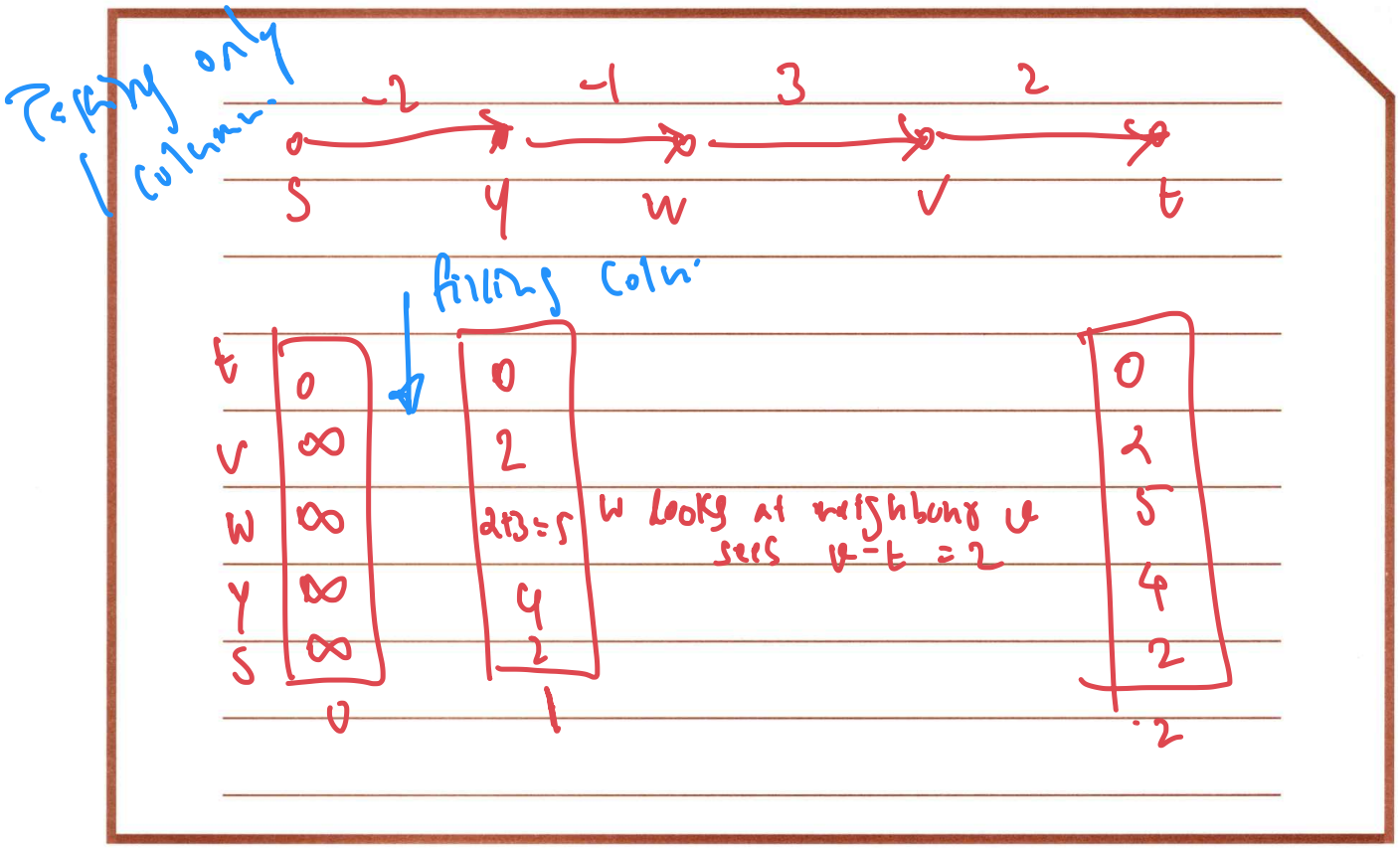
y-t 3

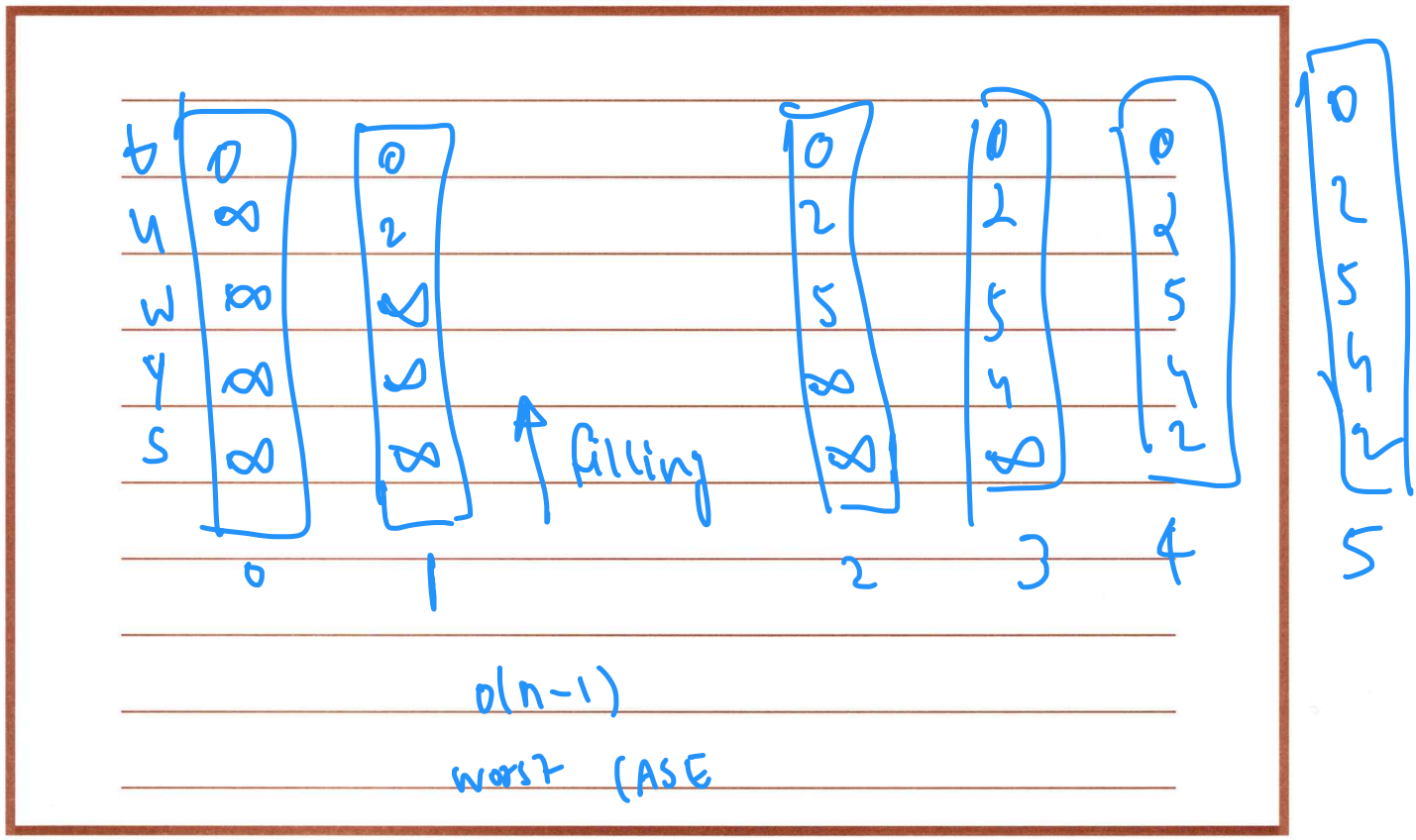
w-t 2

v-t 4

s-t 2

At any time atmost only last 2 col's are required.





What is the preference of filling

BFS order from T

As its neighbours will convey info then their neighbours

:

So on

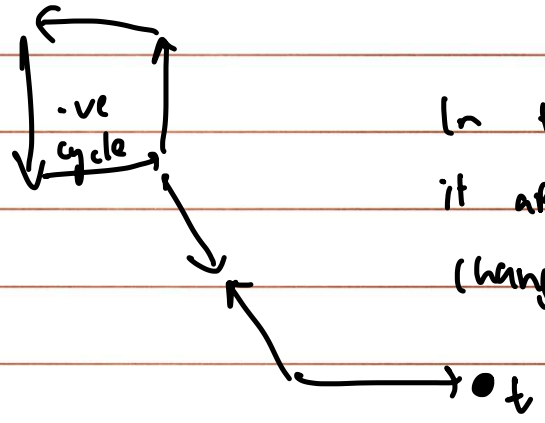
We found shortest distance

For shortest path:

we just keep track of parent i.e neighbour that give shortest distance to t.

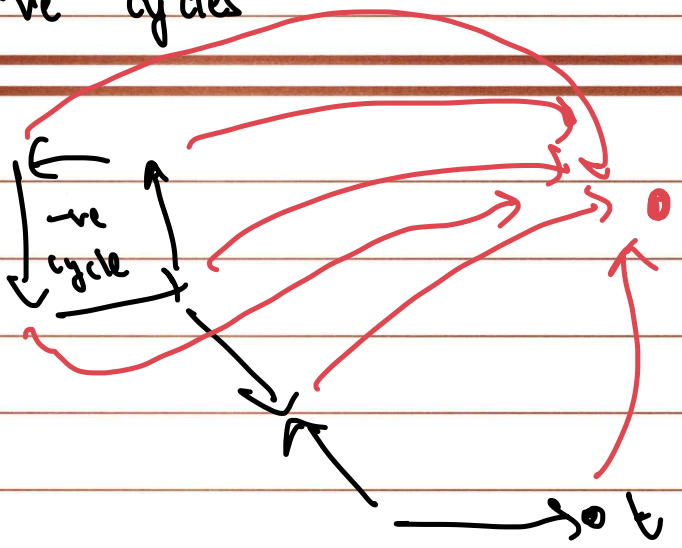
∴ shortest path always found in n-1 iterations

(*) if after n-1 iterations value go down this indicates presence of -ve cycle



In this case running it after n-1 times wont change col^n value as -ve cycle is not on path to t

Then how to find node from where to run BellmanFord and check if there are -ve cycles



Add new node t^* Connect every node to t^*

G'

Now run Bellman Ford for t^*

Bellman Ford

$O(m \cdot n)$

If no -ve cost edges can use BF | Dijkstra's

Dijkstra's

$O(n \log n)$

If -ve cost edges use Bellman ford

Discussion 7

1. When their respective sport is not in season, USC's student-athletes are very involved in their community, helping people and spreading goodwill for the school. Unfortunately, NCAA regulations limit each student-athlete to at most one community service project per semester, so the athletic department is not always able to help every deserving charity. For the upcoming semester, we have S student-athletes who want to volunteer their time, and B buses to help get them between campus and the location of their volunteering. There are F projects under consideration; project i requires s_i student-athletes and b_i buses to accomplish, and will generate $g_i > 0$ units of goodwill for the university. Our goal is to maximize the goodwill generated for the university subject to these constraints. Note that each project must be undertaken entirely or not done at all -- we cannot choose, for example, to do half of project i to get half of g_i goodwill.

2. Suppose you are organizing a company party. The corporation has a hierarchical ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee *j* has a value v_j (a positive integer), representing how enjoyable their presence would be at the party. Our goal is to determine which employees to invite, subject to these constraints, to maximize the total value of invitees.

3. You are given a set of n types of rectangular 3-D boxes, where the i^{th} box has height h(i), width w(i) and depth d(i) (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

In some case Bellman ford works better than Dijkstras as Bellman ford can be easily parallized As Bellman ford only looks at its neighbors but Dijkstras need the whole G ∴ it needs info centrally whereas Bellman ford does not need info centrally.

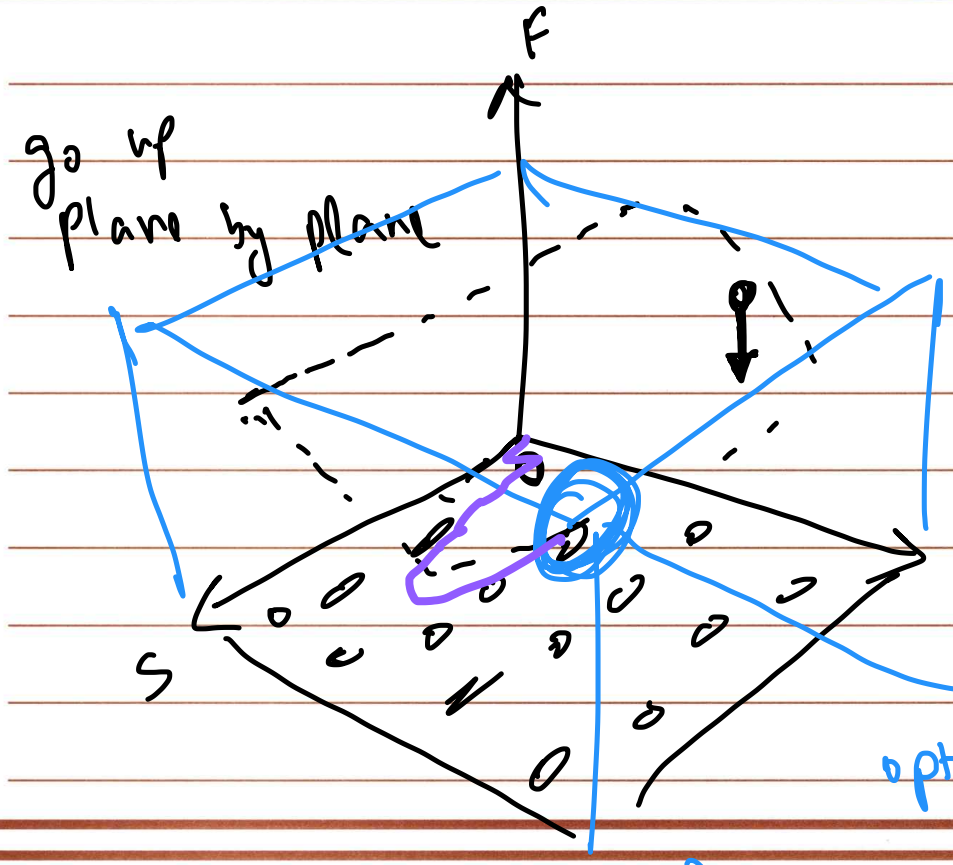
cpu cost — fastest
I/O cost
communication cost — slowest

(1) S - Students
B - bases
F - projects

In 0/1 Knapsack we had 1 resource (W)
Here we have 2 resources (S, B)

OPT(i, S, B) = optimal value of the soln for projs 1 ... i with S students and B bases.

$$OPT(i,S,B) = \max\left(\begin{array}{l} G_i + OPT(i-1, S-S_i, B-B_i), \\ OPT(i-1, S, B) \end{array}\right)$$



$O(B * F * S)$

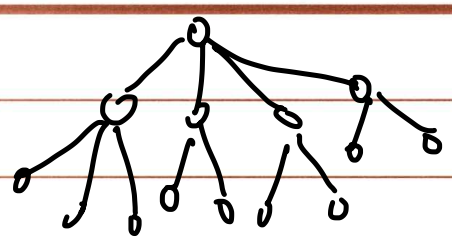
$O(F)$ to complete to Down Parse

not efficient

because of B and S

BFS $\rightarrow$ $F\ 2^{\log_2 B}\ 2^{\log_2 S}$

(2)



$OPT(i)$ = max fun-factor for the subtree rooted at node i

$$OPT(i) = \max\left(V_i + \sum_{g \in \text{grandchildren}(i)} OPT(g) \;,\; \sum_{c \in \text{children}(i)} OPT(c) \right)$$

$O(n)$ Time

③ Box Type height $h_i = 2$ $2 \times (3 \times 5)$

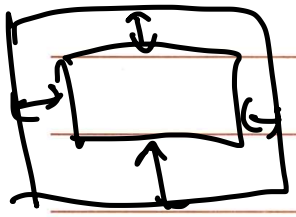
width $w_i = 3$ $3 \times (2 \times 5)$

Depth $d_i = 5$ $5 \times (2 \times 3)$

↑ height ↑ depth ↑ width

Instead of n boxes

we can consider $3 * n$ boxes that cannot rotate



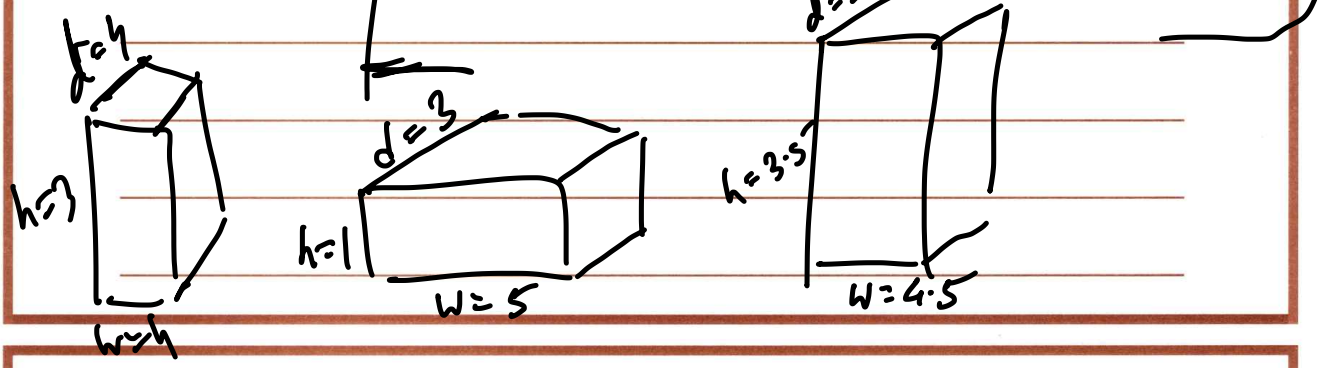
Can sort boxes based on Base Area, depth, width.

Suppose we sort acc. to Base Area

$H(j)$ = Height of tallest stack of boxes $i \cdots j$

$$H(j) = \max\left(H(j-1),\ \max\left(h_j + H(k) \right) \right)$$

$i \le k < j$ AND $w_k > w_j$ AND $d_k > d_j$



$H(1) = 3$

$H(2) = 3$

$H(3) = 3.5 + 3 = 6.5$

h_j $H(3)$ came from $H(2)$

$H(2)$ came from $H(1)$

Box 3 compatible with Box 2

Box 3 and Box 1 not compatible ∴ above formula is wrong

$H(j)$ = Height of tallest stack of boxes $1 \cdots j$ with j at the top of stack.

$$H(j) = h_j + \max_{\substack{i<j \text{ AND} \\ w_j < w_i \text{ AND} \\ d_j < d_i}} \left[H(i) \right]$$

$H()$

Need to search for highest stack where we are done filling out H array.

Takes $O(n^2)$