Dynamic Programming

Sequence Alignment Problem

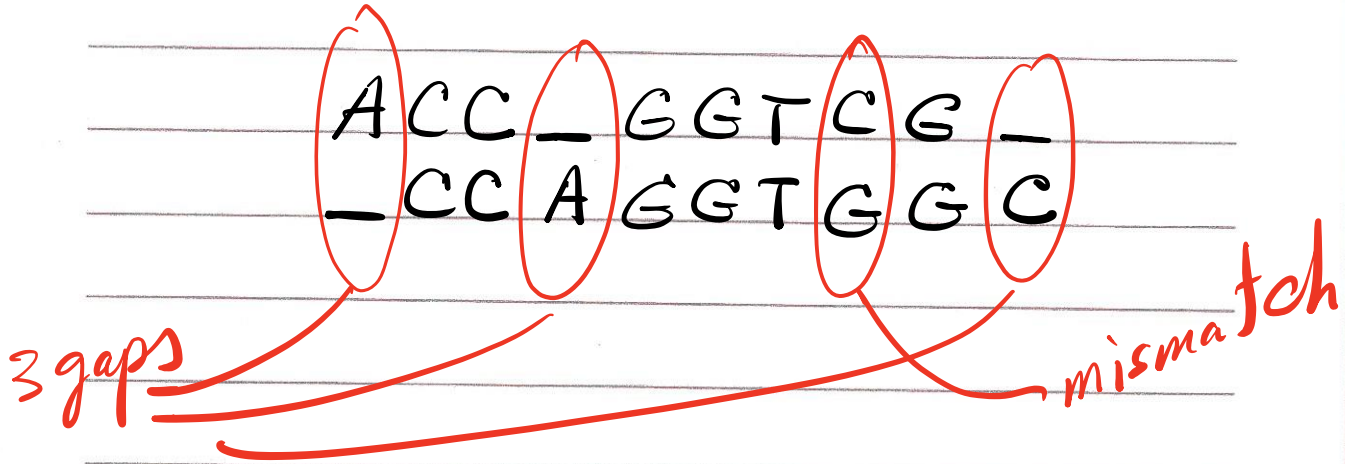
A DNA strand consists of a string of molecules called bases!

4 Types of bases:

- Adenine A
- Cytosine C
- Guanine G
- Thymine T

$S_1 = ACCGGTCG$

$S_2 = CCAGGTGGC$



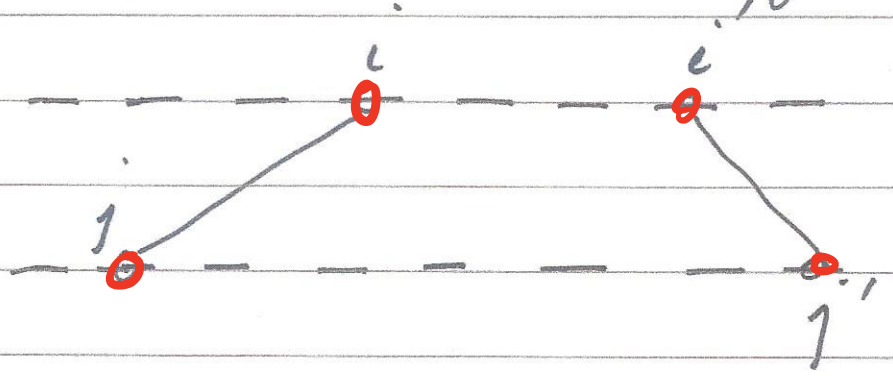
Suppose we have 2 strings X & Y.

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

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

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

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



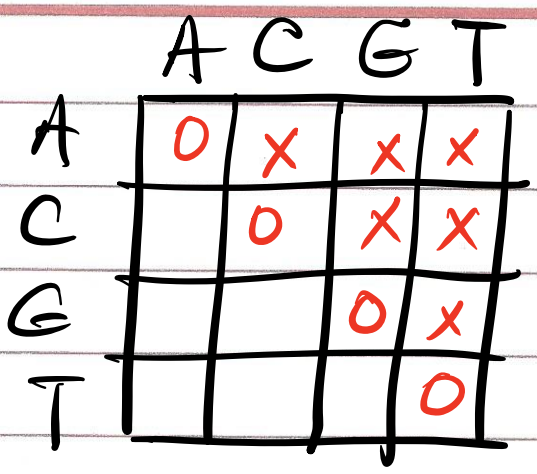
$(i, j), (i', j') \in M$

& $i < i' \rightarrow j < j'$

For an alignment M between X & Y

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

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



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

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

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

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

Define $OPT(i, j)$ as the min cost of an alignment between $x_1 \ldots x_i$ & $y_1 \ldots y_j$

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

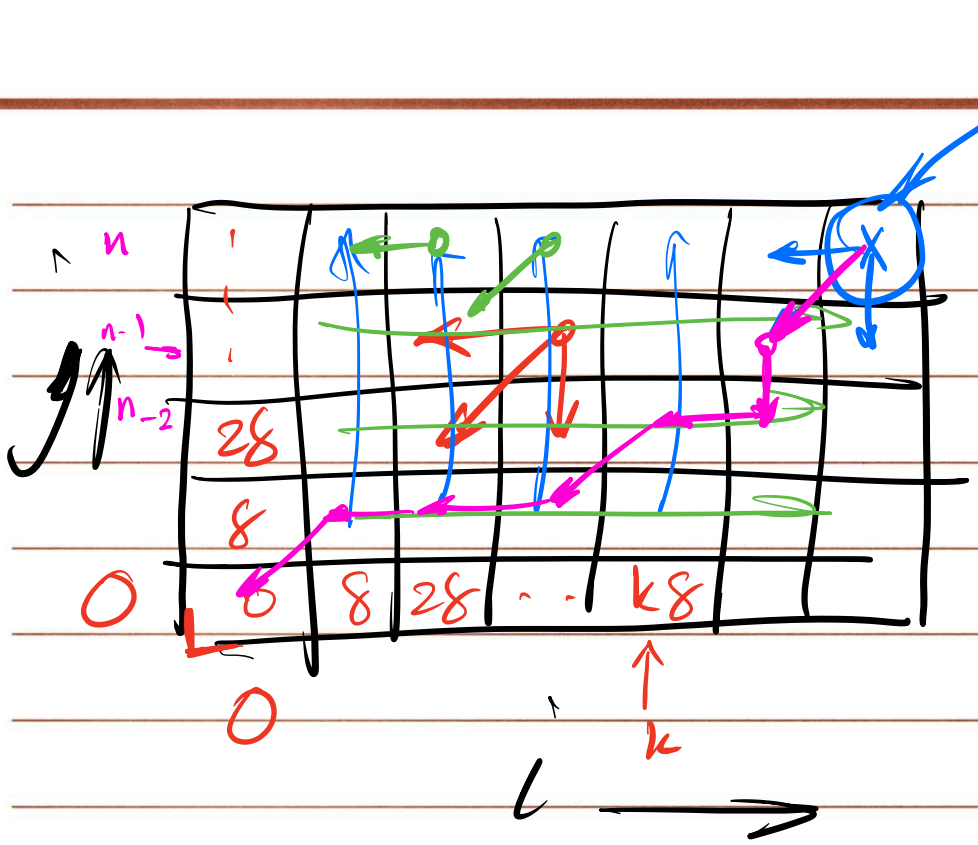
1) - $(x_m, y_n) \in M \Rightarrow OPT(m,n) = OPT(m-1, n-1) + \alpha_{x_m y_n}$

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

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

$$OPT(i,j) = \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1), \\ \delta + OPT(i-1, j), \\ \delta + OPT(i, j-1) \end{cases}$$

rec. formula

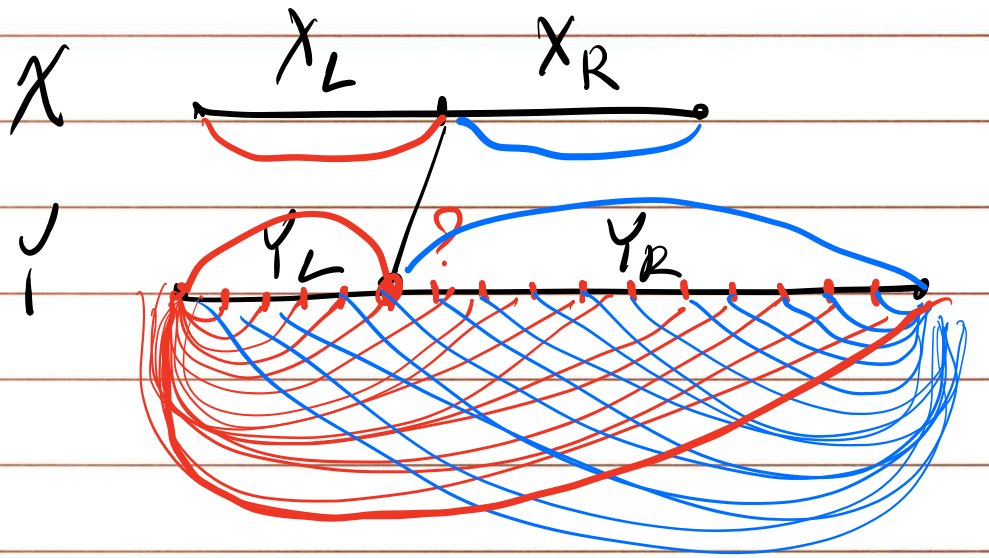


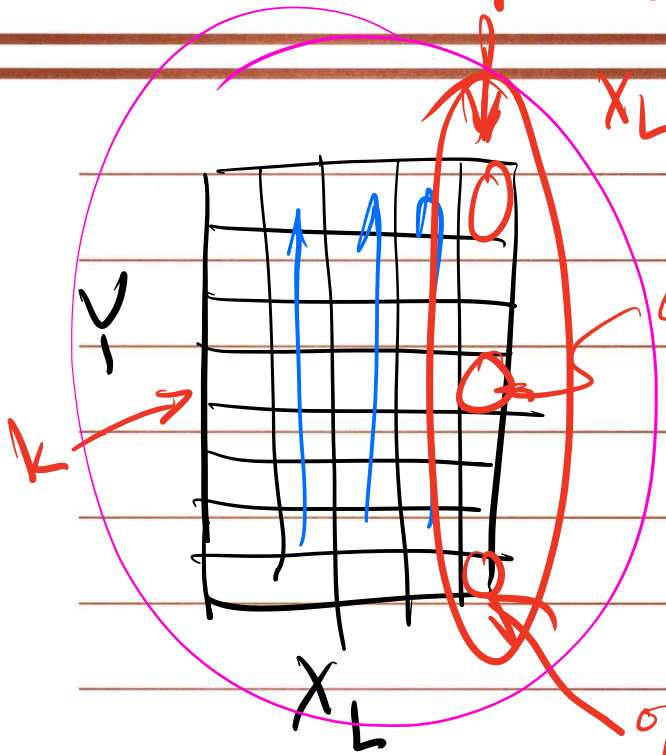
Initialize Col. 0 & Row 0 as above

```
for i = 1 to m
    for j = 1 to n
        use rec. formula (1)
    endfor
endfor
```

O(n) O(n)

takes O(mn)

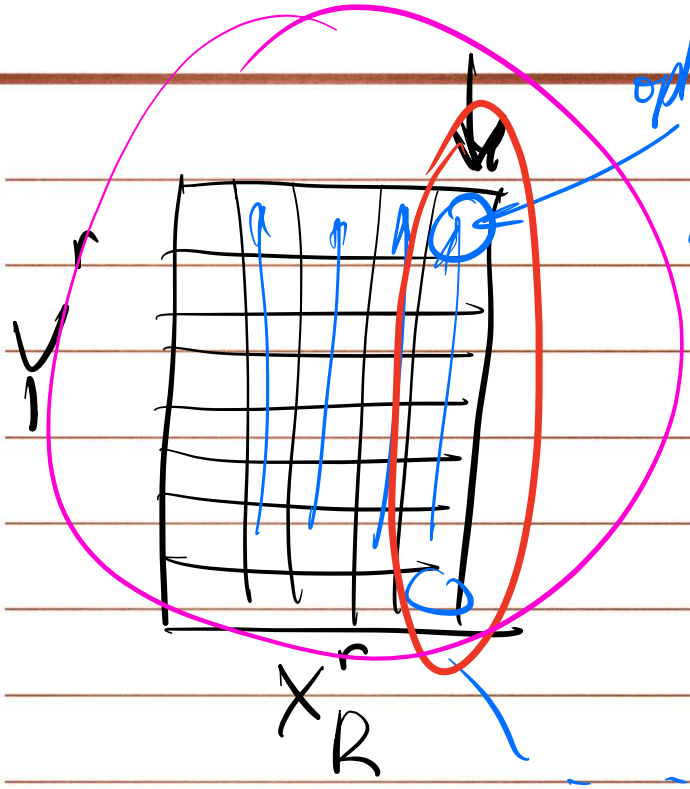




opt. cost of an alignment between X_L and all of Y.

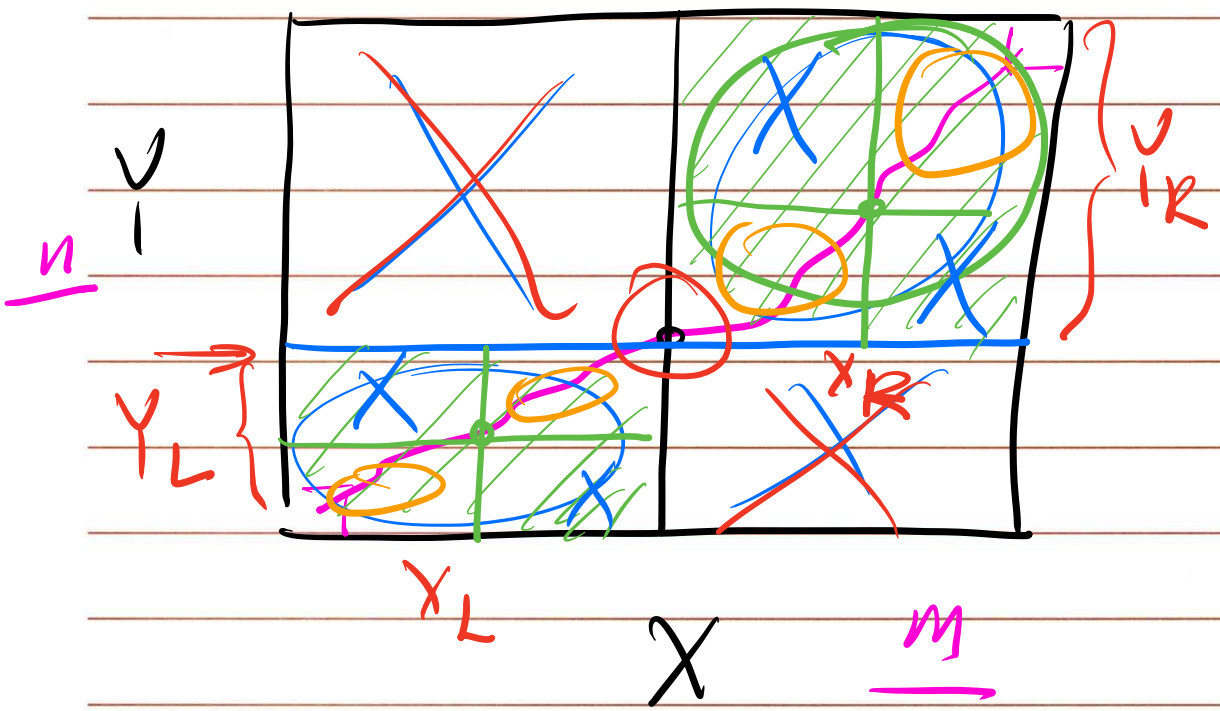
opt. cost of an align. between X_L and $Y_1 \dots Y_k$

opt. cost of an align. between X_L and none of Y.



opt. cost of alignment between X_R and all of Y

- - - none of Y.



root level → Cmn

$\frac{1}{2}Cmn$

$\frac{1}{4}Cmn$

$\sum = 2Cmn = O(mn)$

Matrix Chain Multiplication

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

$$m\underbrace{[\ A\ ]}_{n}\underbrace{[\ B\ ]}_{k}\}n = [\ C\ ]$$

total # of op's $= m \times n \times k$

$B \cdot C \cdot D$

$\begin{cases} B \text{ is } 2 \times 10 \\ C \text{ " } 10 \times 50 \\ D \text{ " } 50 \times 20 \end{cases}$

$B \cdot (CD) = 10{,}400$

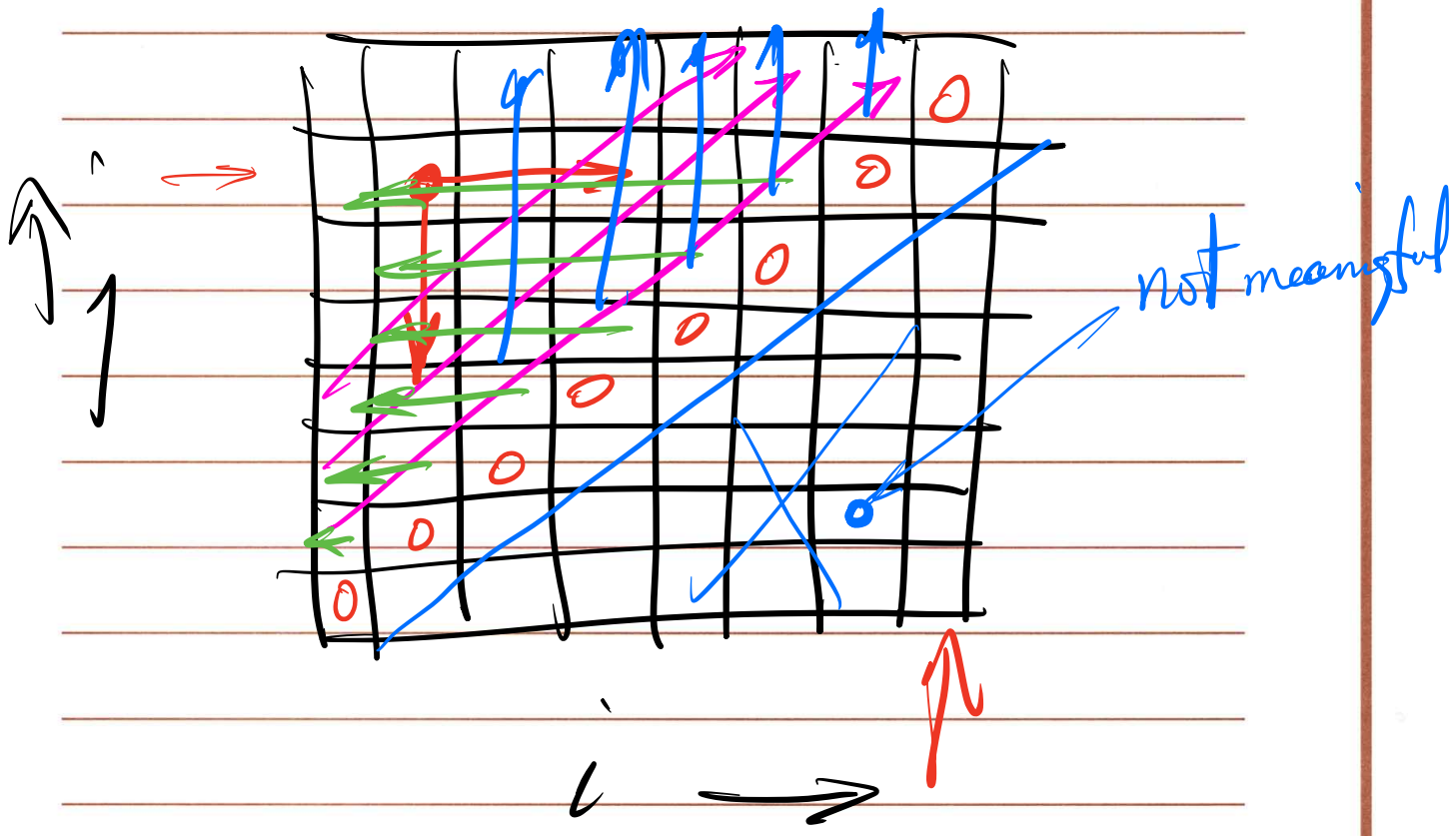
$(BC) D = 3{,}000$

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

$OPT(i,j)$ = opt. cost of multiplying matrices $A_i \cdots A_j$

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

2



for $i=1$ to n, $OPT(i,i)=0$

$O(n)$ (for $j=2$ to n

  $O(n)$ (for $i=j-1$ to 1 by -1

    use rec. formula (2)

    $\uparrow$ $O(1)$

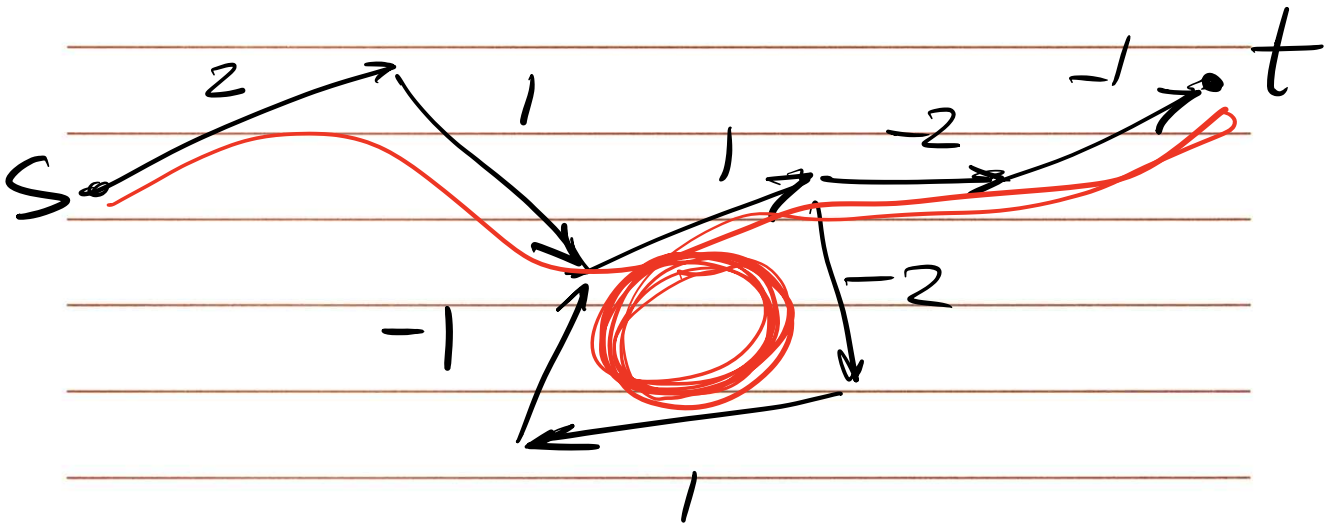
  endfor

endfor)

$\rightarrow$ takes $O(n^3)$

Shortest Path Problem

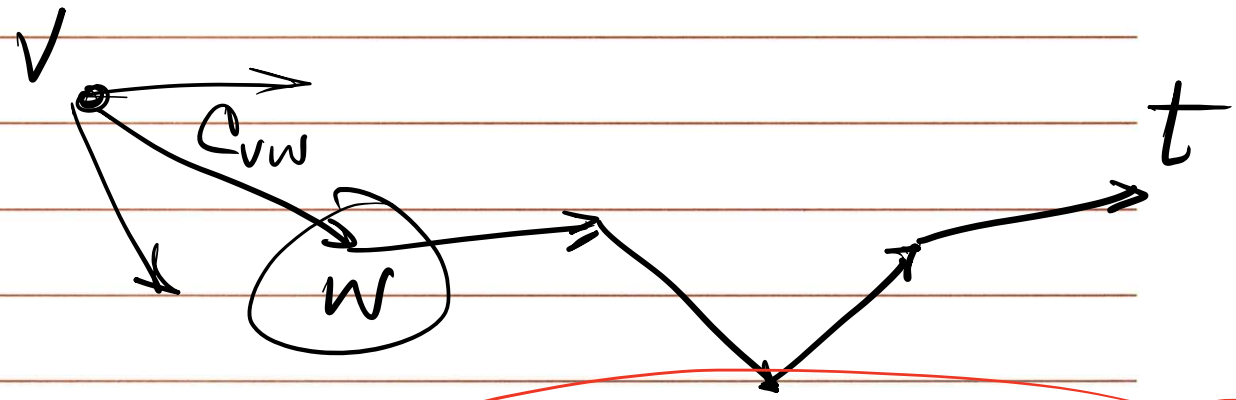
Dynamic Programming



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

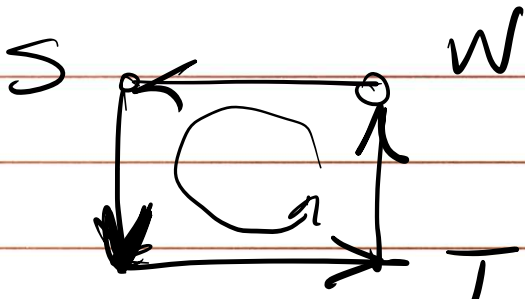
→ OPT(i, v) denotes the min cost of a v-t path using at most i edges

objective: find OPT(n-1, s)

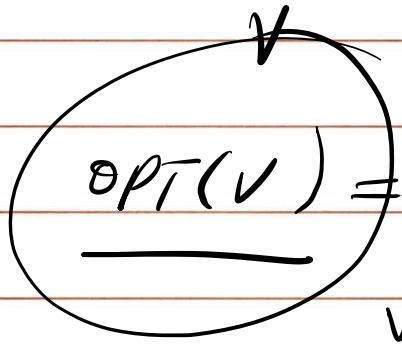


$$OPT(i,v) = \min_{w \in Adj(v)} \left(OPT(i-1,w) + c_{vw} \right)$$

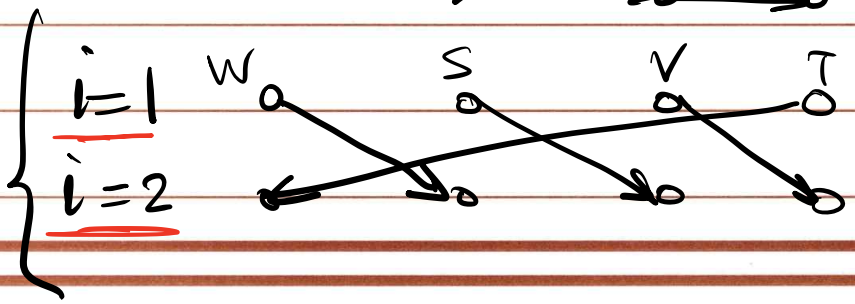
$$OPT(i,v) = \min \left(OPT(i-1,v), \min_{w \in Adj(v)} \left(OPT(i-1,w) + c_{vw} \right) \right)$$

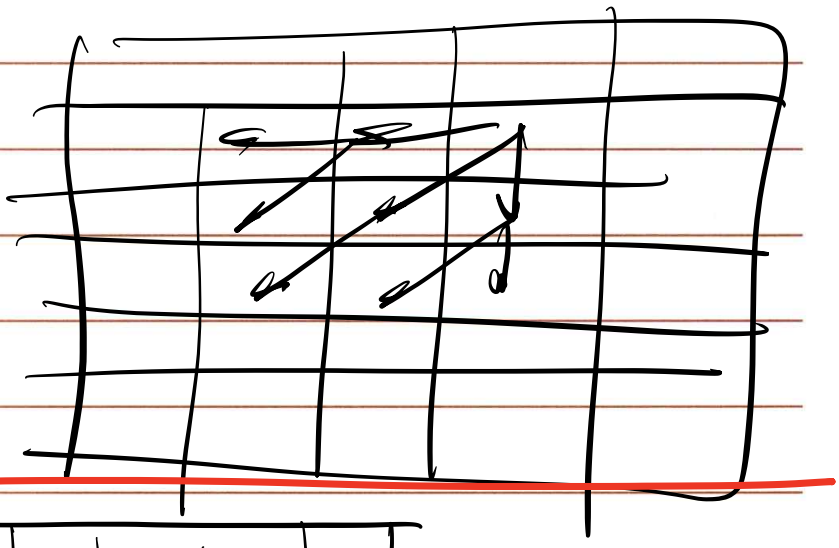


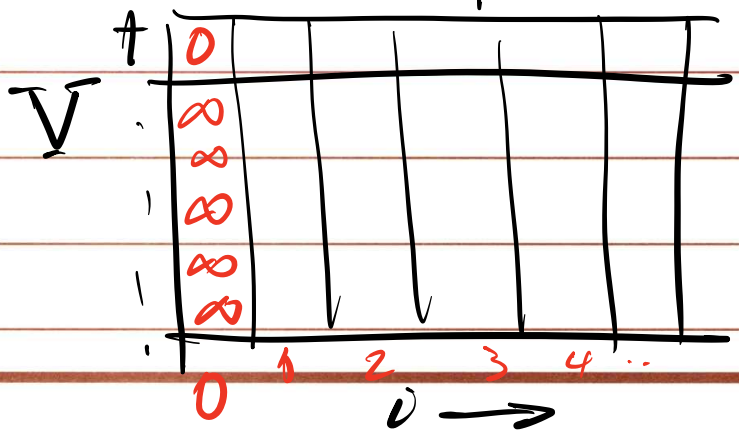
$OPT(T) = 0$



$OPT(V)$ = shortest distance from V to t







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 $n-1$

  for $v \in V$ in any order

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

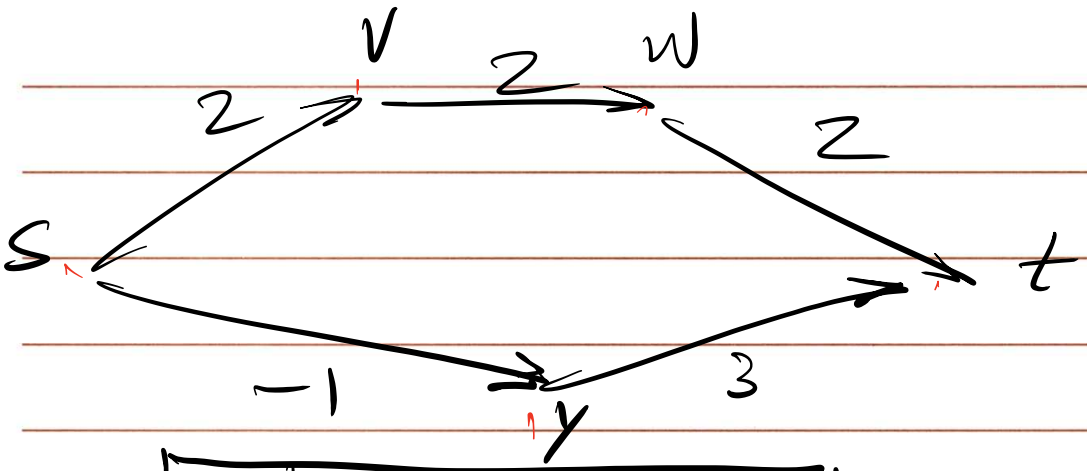
  end for

end for

$O(n)$ $O(n)$ $O(n)$

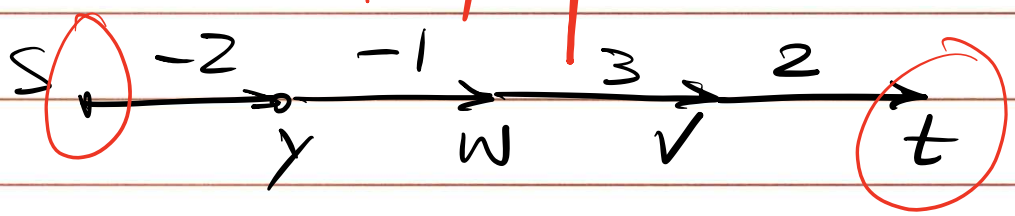
Takes $O(n^3)$

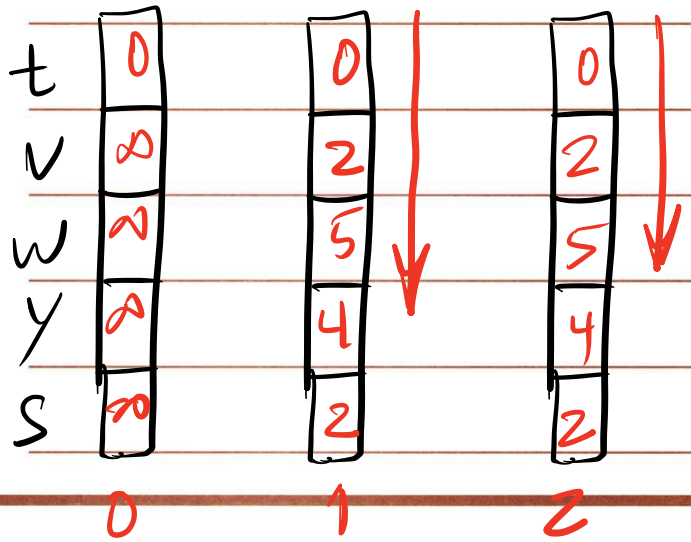
↳ can be reduced to $O(mn)$

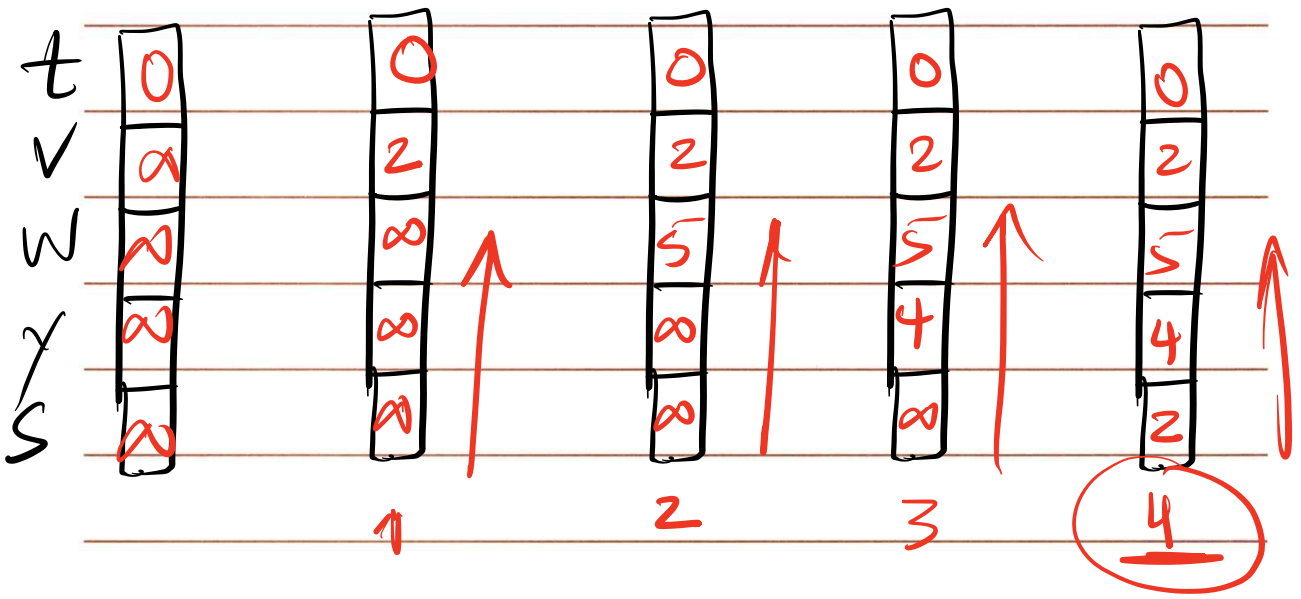


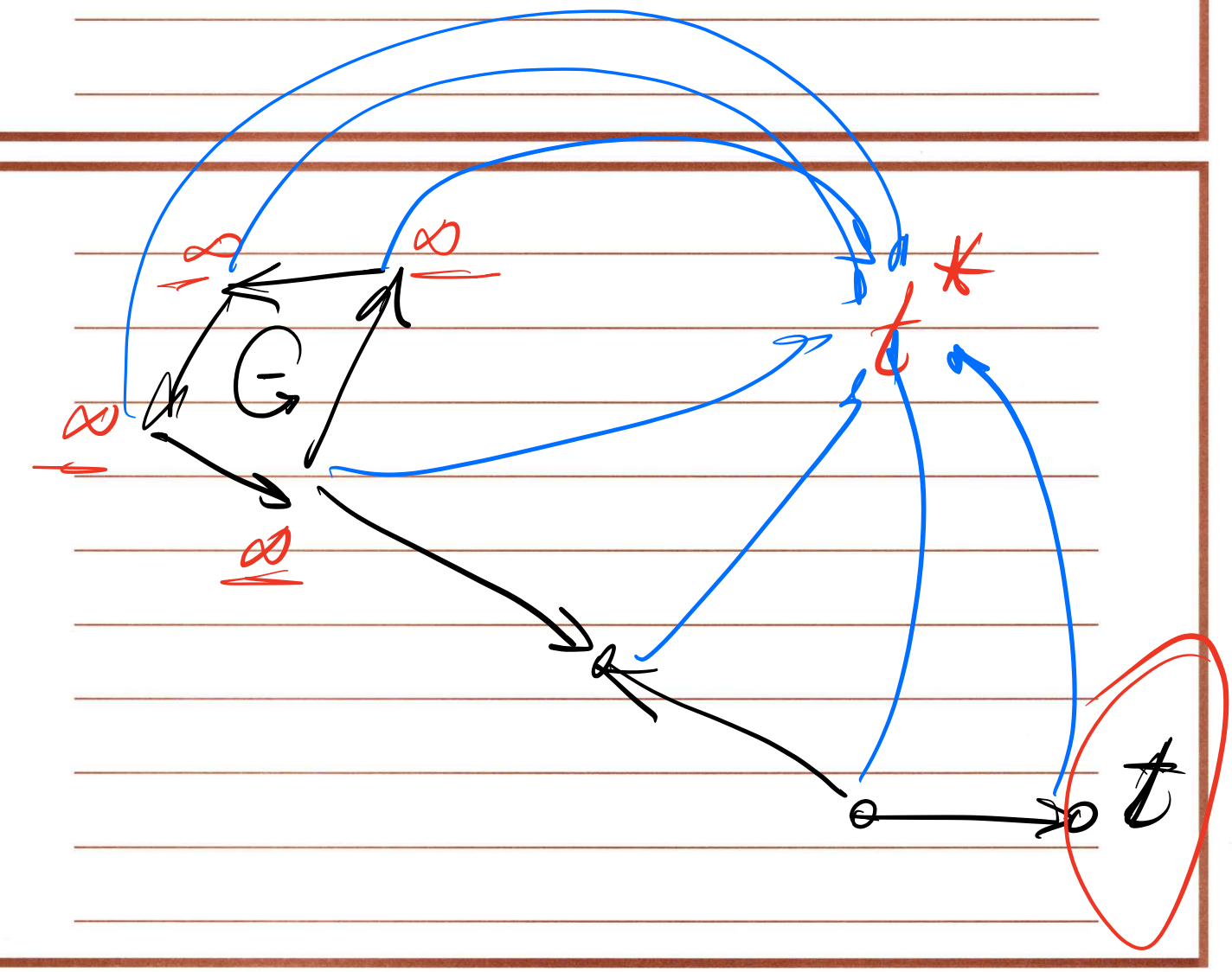
t	0	0	0	0	0	
y	∞	3	3	3		
v	∞	∞	4	4		
w	∞	2	2	2		
s	∞	∞	2	2		

0 1 2 3 4 5






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


∞
∞
G
∞
∞
t
*
t

Bellman Ford	Dijkstra's
$O(mn)$	$O(m \log n)$
CPU operations	least expensive
I/O "	
Message passing "	most expensive

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.

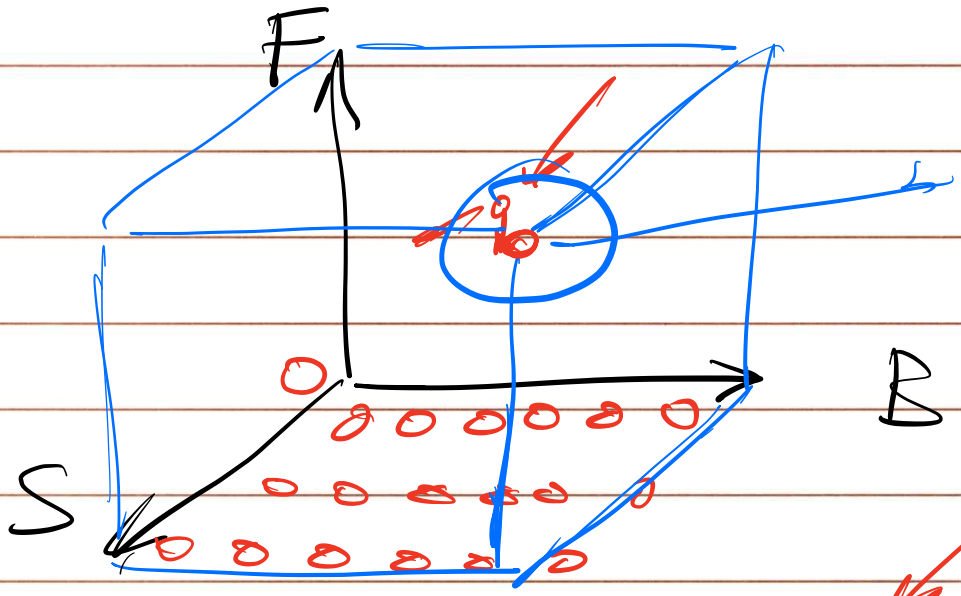
1. When their respective sport is not in season, USC's student-athletes are very involved in their community, helping people and spreading goodwill for the school. Unfortunately, NCAA regulations limit each student-athlete to at most one community service project per semester, so the athletic department 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.

0-1 knapsack

OPT(i, w) = opt. value of the sol. w/ req's 1..i and cap w.

OPT(i, S, B) = opt. value of the sol. for proj's 1..i w/ S students & B buses

$$OPT(i, B, S) = \max(g_i + OPT(i-1, S-s_i, B-b_i), OPT(i-1, S, B))$$



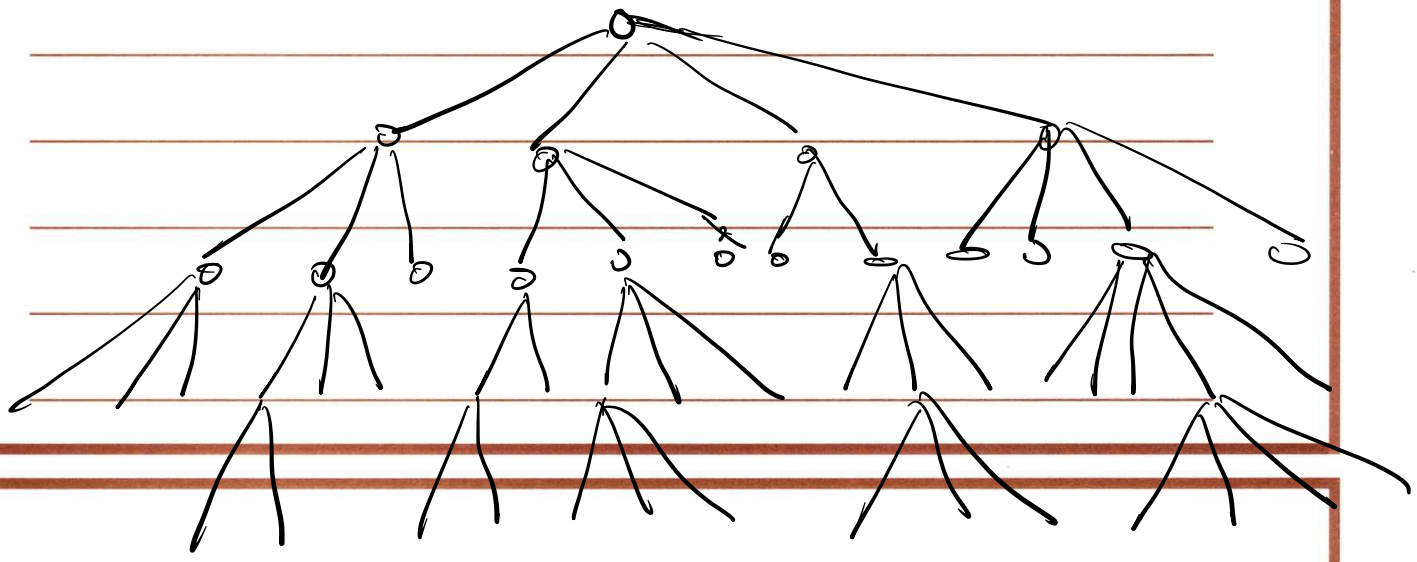
value of the opt. Sol.

Takes $O(BFS)$

pseudopolynomial complexity.

Top down pass takes $O(F)$

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.



$OPT(i)$ = Max. fun factor for subtree rooted at node i

$$OPT(i) = \max\left(V_i + \sum_{g \in g_i} OPT(g),\ \sum_{c \in C_i} OPT(c)\right)$$

Can be done in $O(n)$

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.

height depth width

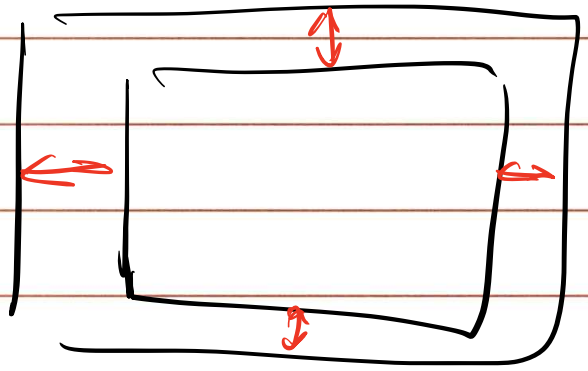
box type: $h_i = 2$, $w_i = 3$, $d_i = 5$

$2 \times (3 \times 5)$

$3 \times (2 \times 5)$

$5 \times (2 \times 3)$

n box types ⟶ 3n boxes.

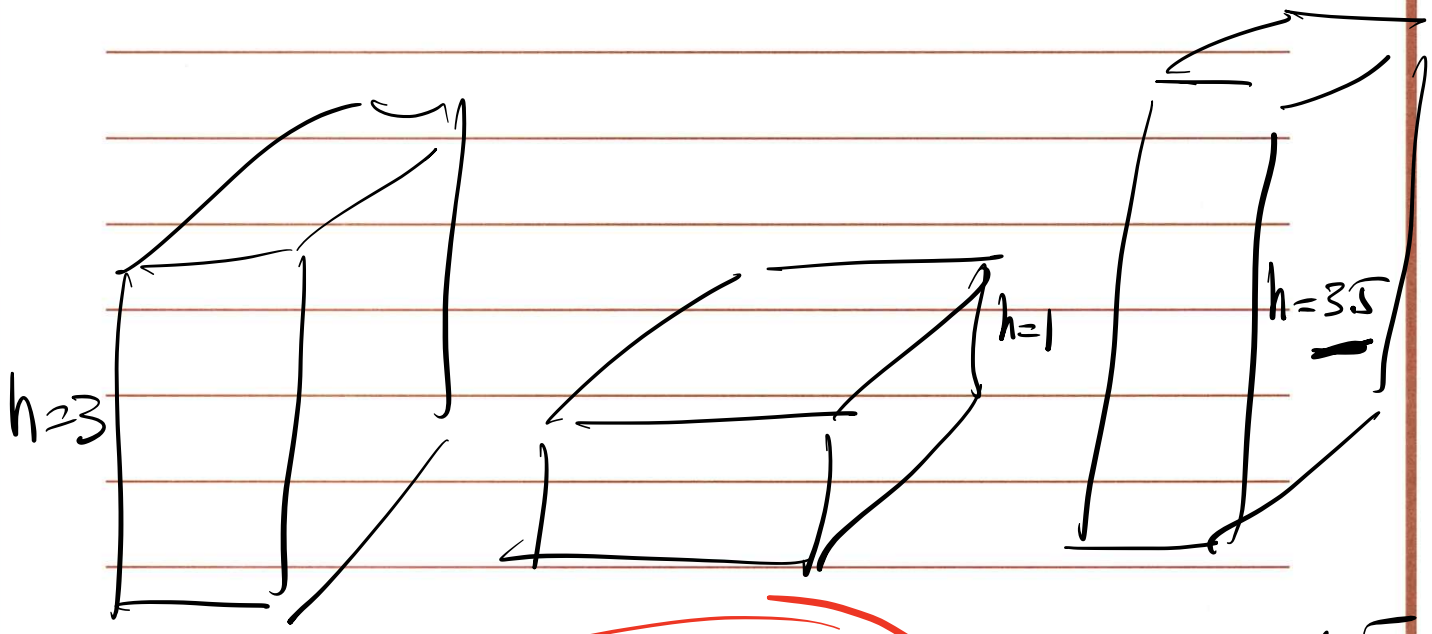


Sort boxes in decreasing order of base area.


OPT(i) = . . .

$H(j)$ = height of the tallest stack of boxes with j

$$H(j) = \max\left[H(j-1),\ h_j + \max_{1 \le k < j \ \&\ w_k > w_j \ \&\ d_k > d_j} (H(k)) \right]$$



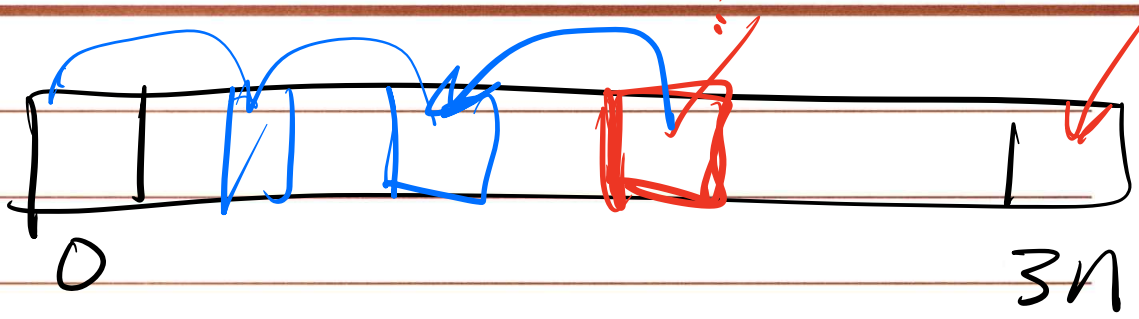
$H(1) = 3$

$H(2) = 3$

$H(3) = 6.5$

$H(j)$ = Height of the tallest stack of boxes $1..j$ w/ j at the top of the stack

$$H(j) = h_j + \max_{i<j \;\&\; w_i<w_j \;\&\; d_i<d_j} [H(i)]$$



This takes $O(n^2)$