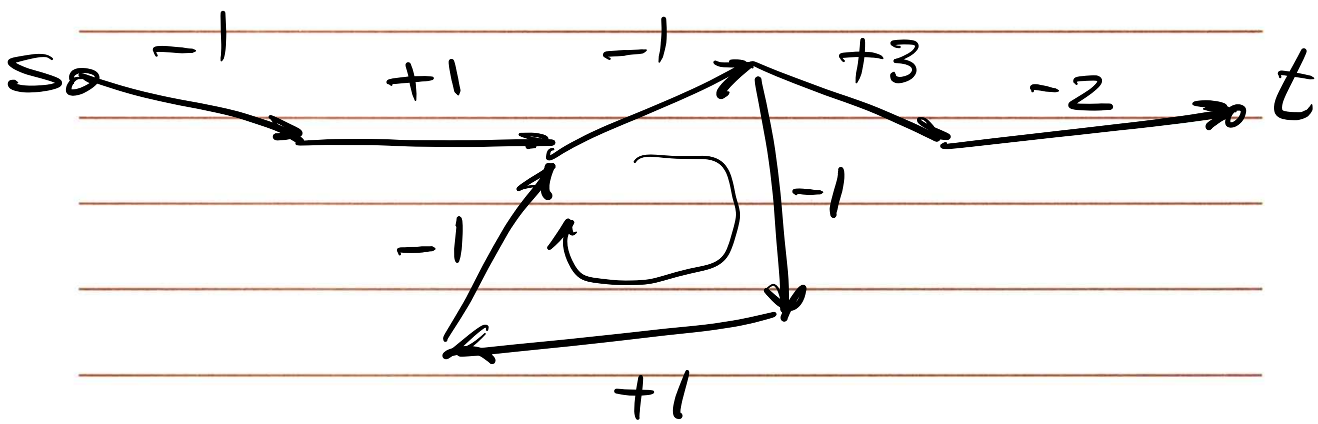
Shortest Path Problem

Dynamic Programming

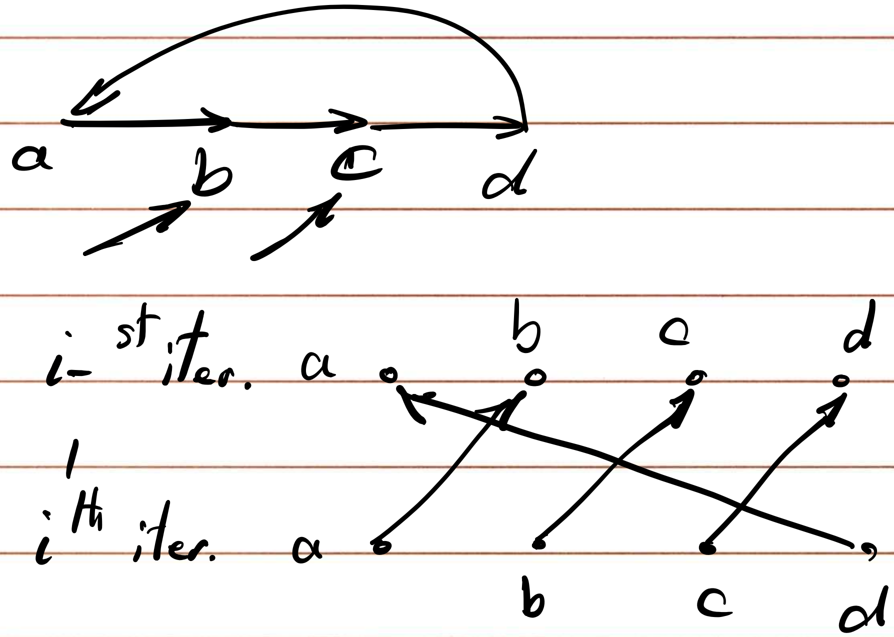


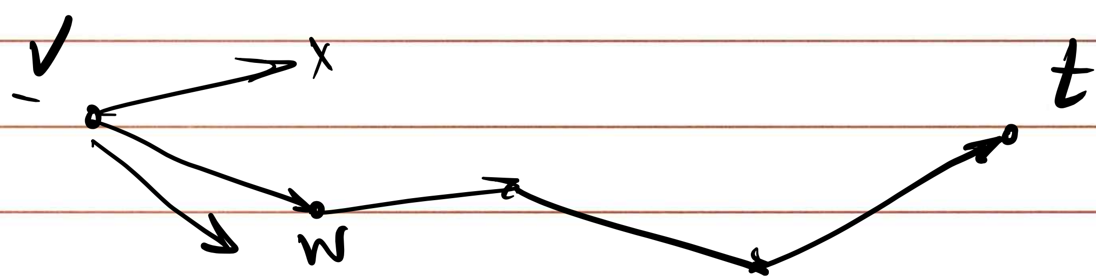
If G has no negative cycles, then there is a shortest path from s to t that is simple and hence has at most n-1 edges.

↓

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

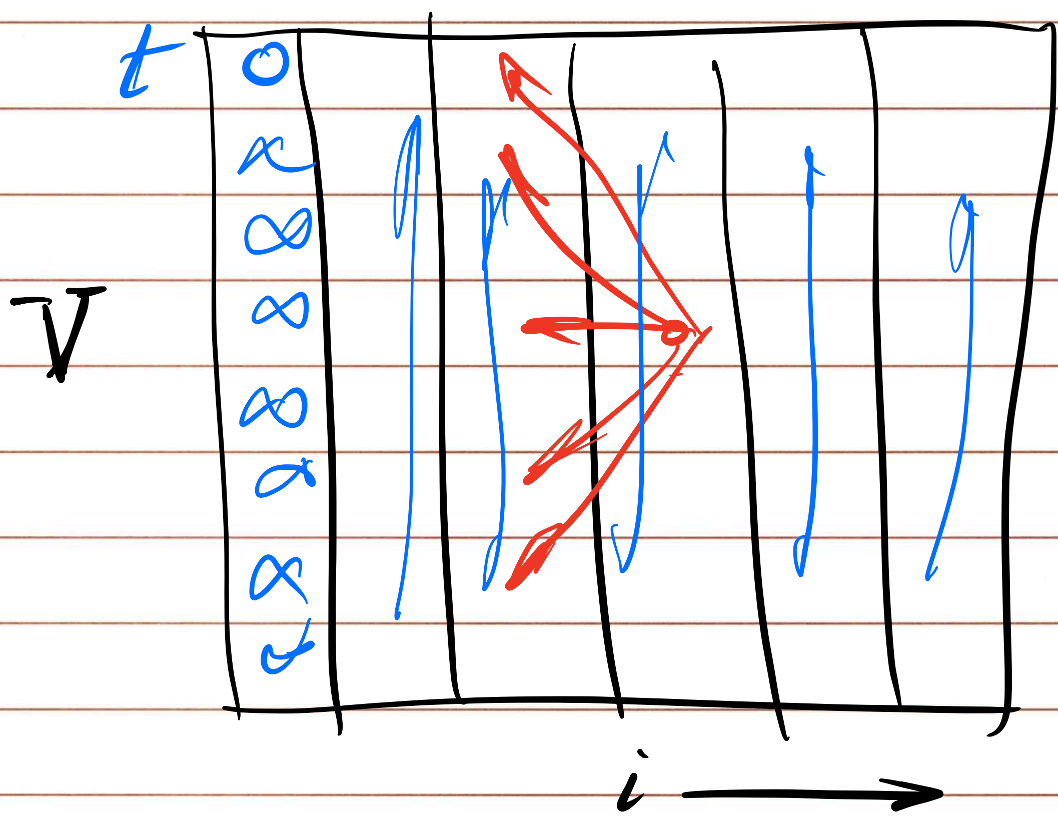
Objective: find OPT(n-1, s)





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

$$OPT(i,v) = \min\Big(OPT(i-1,v),\ \min_{w \in Adj(v)} \big(OPT(i-1,w) + C_{vw} \big) \Big)$$



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 $O(n)$

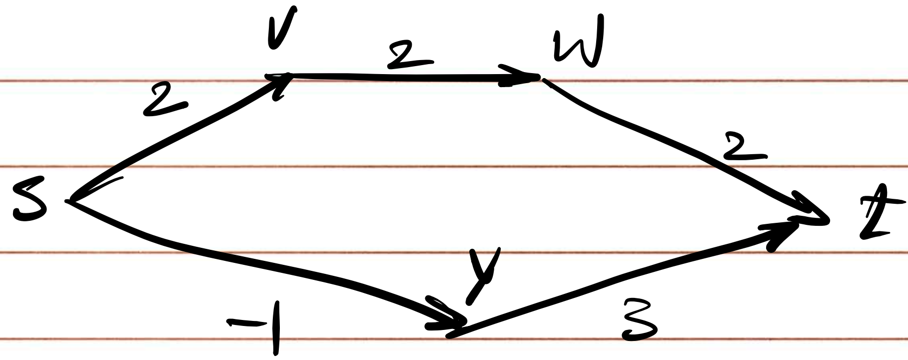
$M[i,v] = \min\left(M[i-1,v], \min_{w \in Adj(v)}\left(M[i-1,w] + C_{vw}\right)\right)$

end for

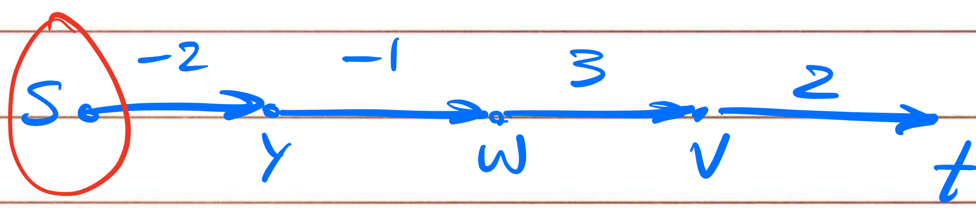
end for

takes $O(n^3)$

$O(mn)$

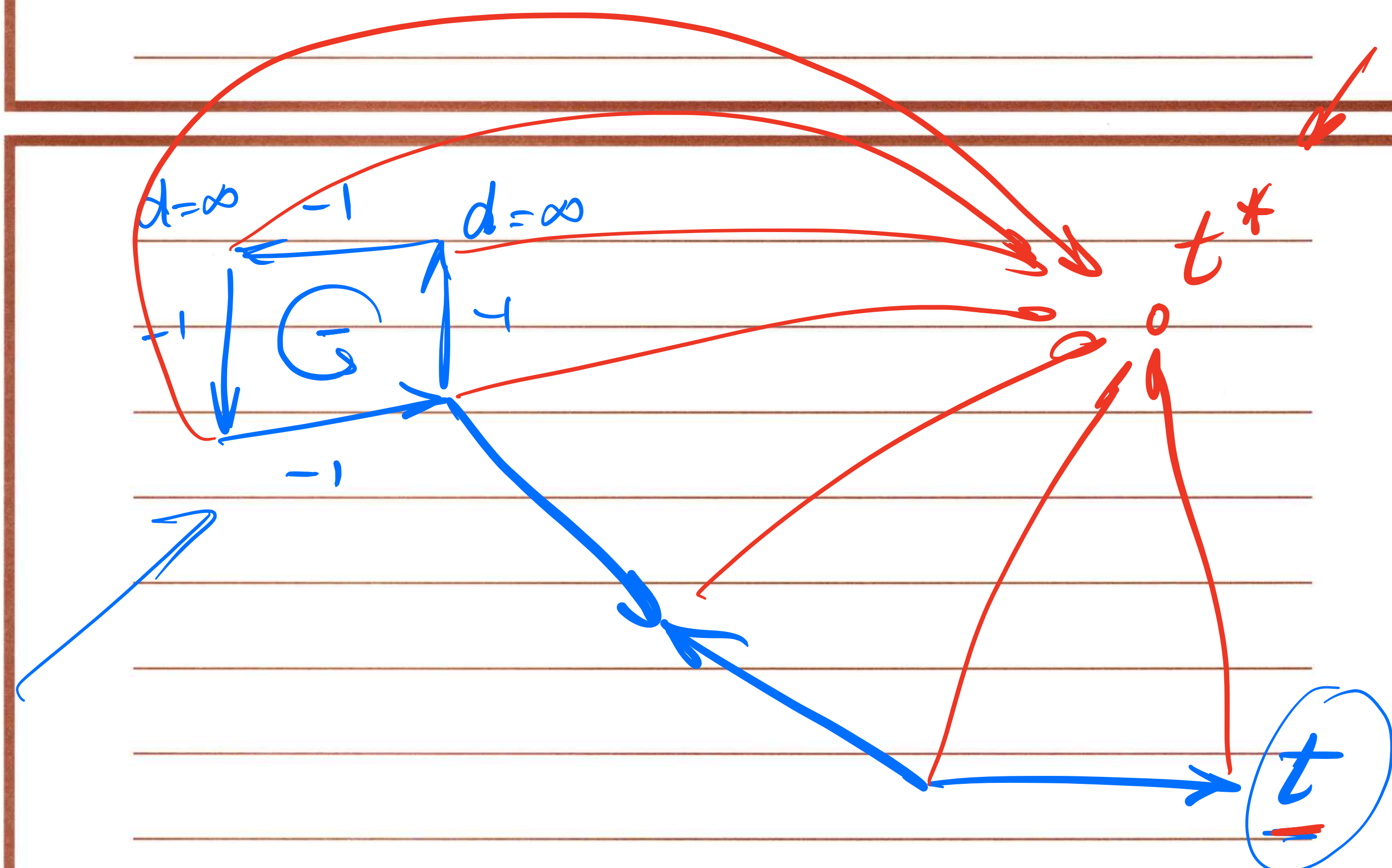


	0	1	2	3	4 ?
t	0	0	0	0	
y	∞	3	3	3	
v	∞	∞	4	4	
w	∞	2	2	2	
s	∞	∞	2	2	



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

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



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

CPU operations — fastest. (least exp.)

I/O "

message passing " — slowest (most exp.)

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.

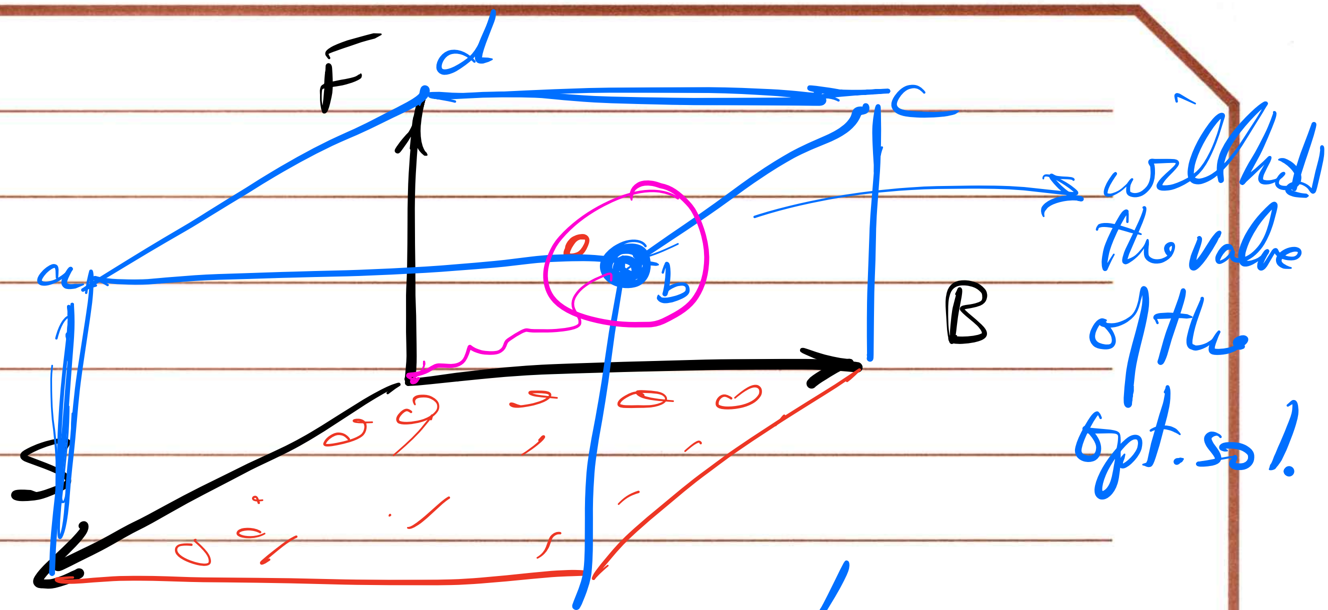
1. When their respective sport is not in season, USC's student-athletes are very involved in their community, helping people and spreading goodwill for the school. Unfortunately, NCAA regulations limit each student-athlete to at most one community service project per semester, so the athletic department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

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



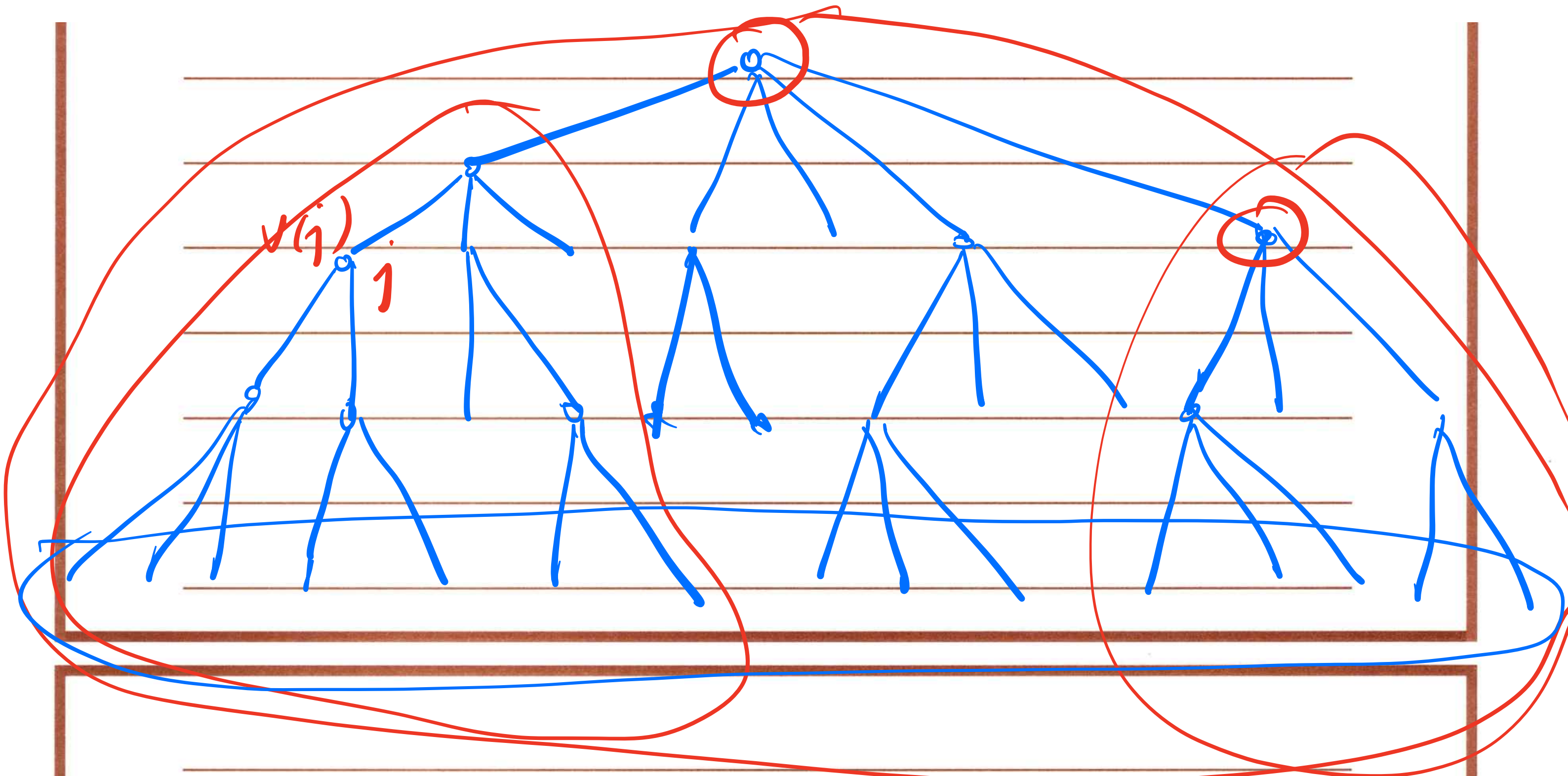
Cost will be: $\Theta(BFS)$

$\Theta(2^{\log_2 B} F 2^{\log_2 S})$

Top down pass will take

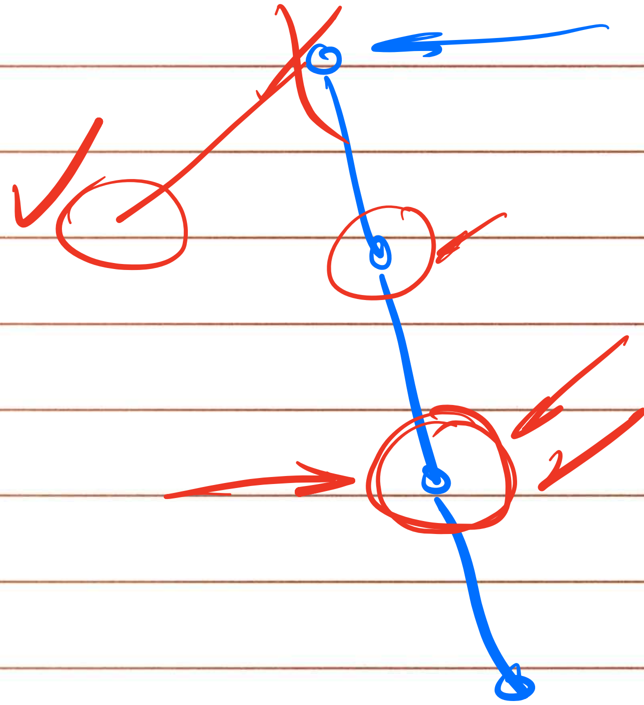
$\Theta(F)$

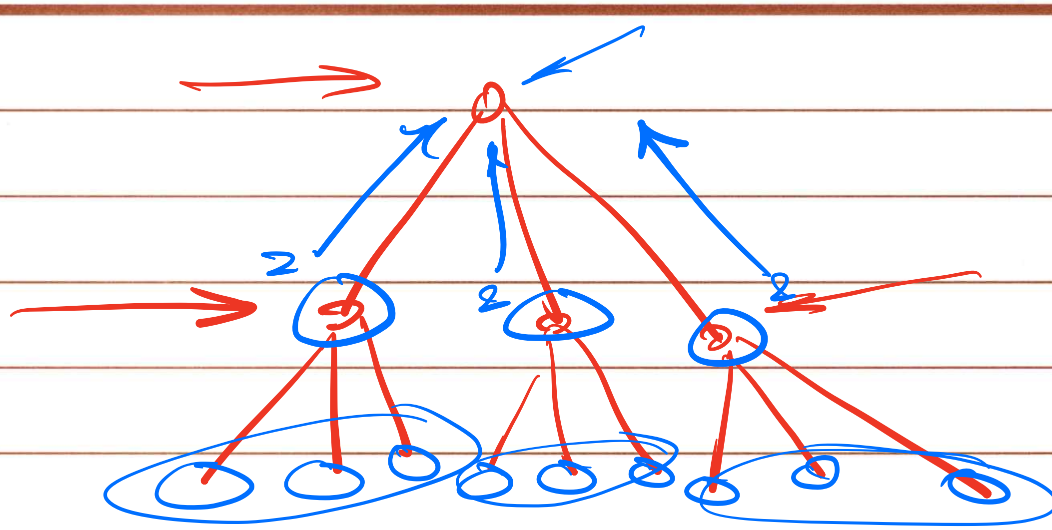
2. Suppose you are organizing a company party. The corporation has a hierarchical ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you 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

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




2
2
2

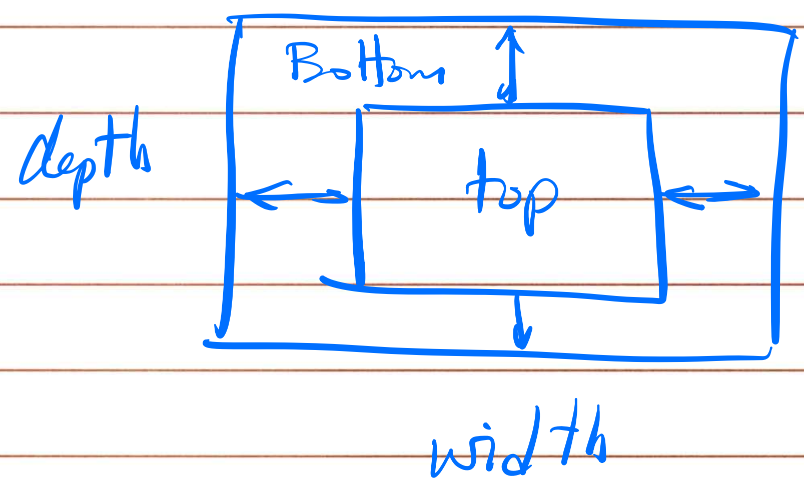
3. You are given a set of n types of rectangular 3-D boxes, where the i^{th} box has height h(i), width w(i) and depth d(i) (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the 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.

box type: $h_i = 2$ → $2 \times (3 \times 5)$

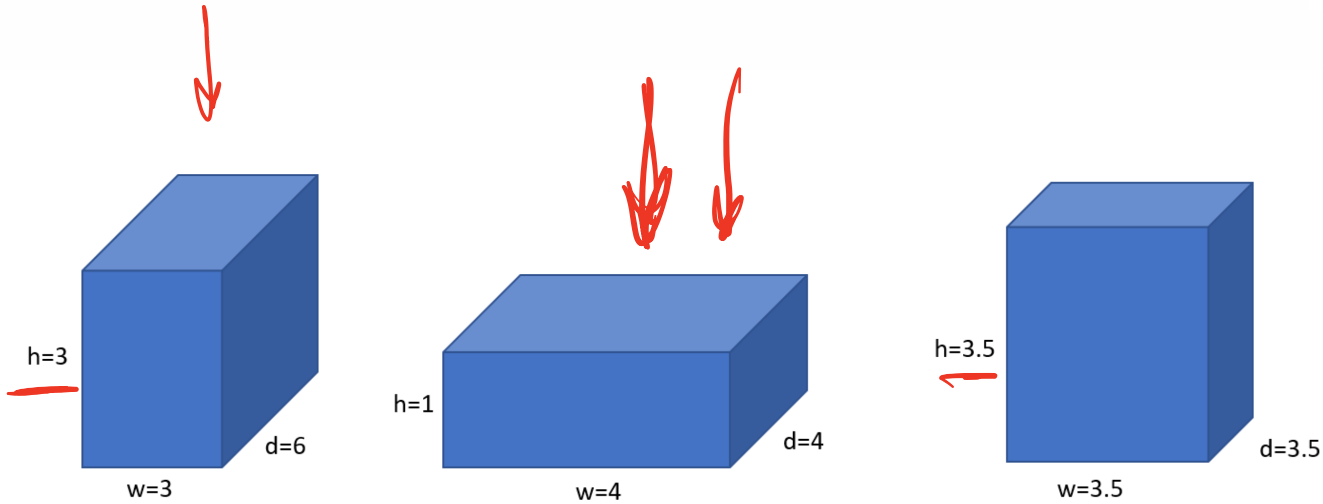
$w_i = 3$ → $3 \times (2 \times 5)$

$d_i = 5$ → $5 \times (2 \times 3)$

depth width



Sort boxes by decreasing order of base area.



1 2 3

OPT(1)=3 OPT(2)=3 OPT(3)= 3.5+3=6.5

$H(j)$ = Height of the tallest stack of boxes 1..j

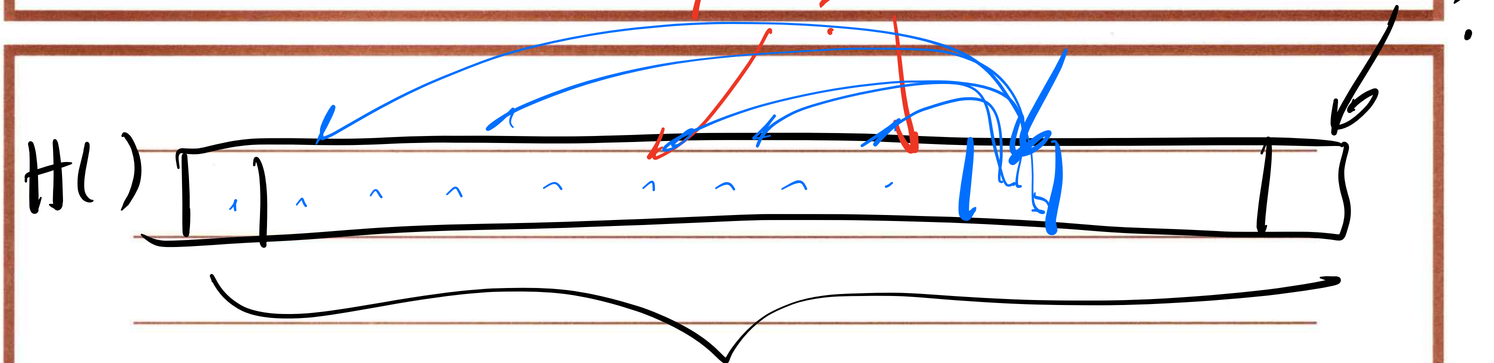
$$H(j) = \max\left[\, H(j-1),\ h_j + \max_{1 \le k < j \ \&\ w_k > w_j \ \&\ d_k > d_j} \left(H(k) \right) \right]$$

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

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

opt. ?

?



$3n$

value of the opt. sol. could be anywhere in $H(\,)$

Cost of the sol. will be $\Theta(n^2)$

Imagine starting with the given decimal number n, and repeatedly chopping off a digit from one end or the other (your choice), until only one digit is left. The square-depth SQD(n) of n is defined to be the maximum number of perfect squares you could observe among all such sequences. For example, SQD(32492) = 3 via the sequence

32492 → **3249** → **324** → 24 → **4**

since 3249, 324, and 4 are perfect squares, and no other sequence of chops gives more than 3 perfect squares. Note that such a sequence may not be unique, e.g.

32492 → **3249** → 249 → **49** → **9**

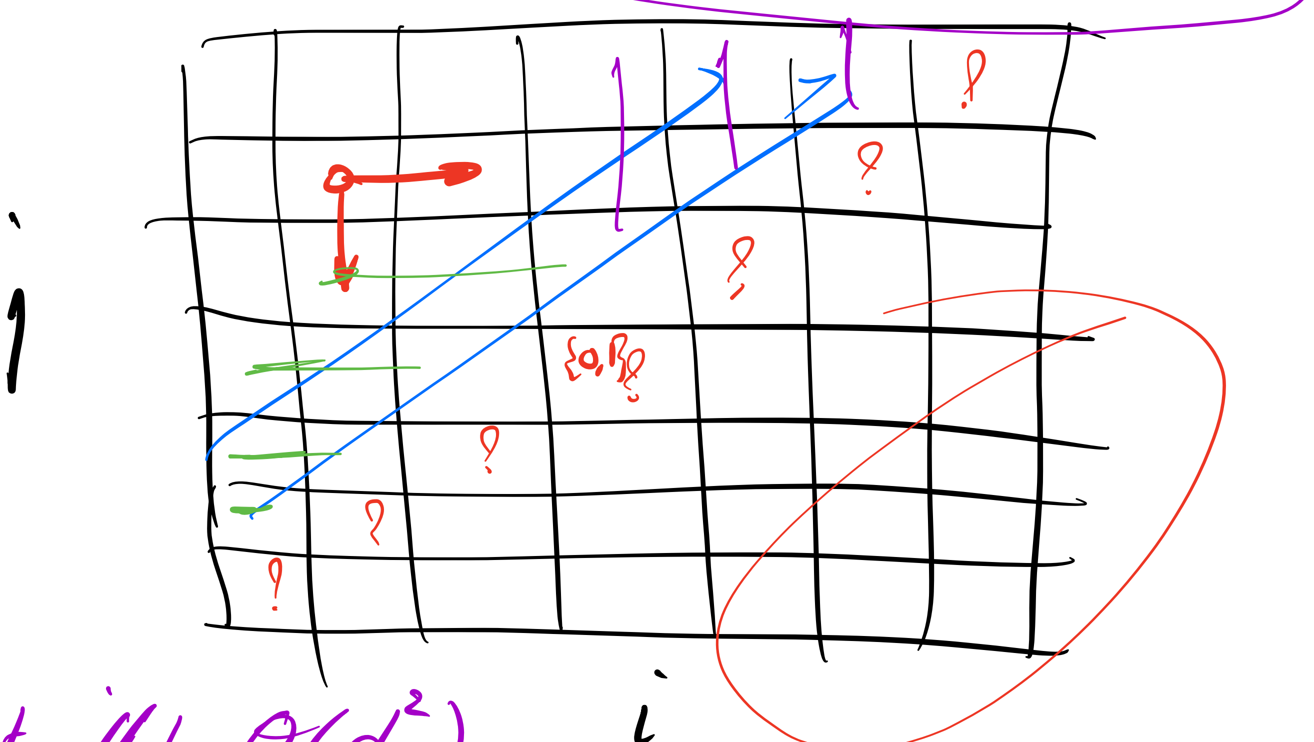
also gives you 3 perfect squares, viz. 3249, 49, and 9.

Describe an efficient algorithm to compute the square-depth SQD(n), of a given number n, written as a d-digit decimal number $a_1 a_2 \ldots a_d$. Analyze your algorithm's running time. Your algorithm should run in time polynomial in d. You may assume the availability of a function IS_SQUARE(x) that runs in constant time and returns 1 if x is a perfect square and 0 otherwise.

OPT(i, j) = Max. no. of squares from digit a_i to digit a_j

$n_{ij} = a_i \ldots a_j$

$OPT(i,j) = \text{is\_Square}(n_{ij}) + \text{Max}(OPT(i+1,j), OPT(i,j-1))$



Cost will be $\Theta(d^2)$