£7 5800

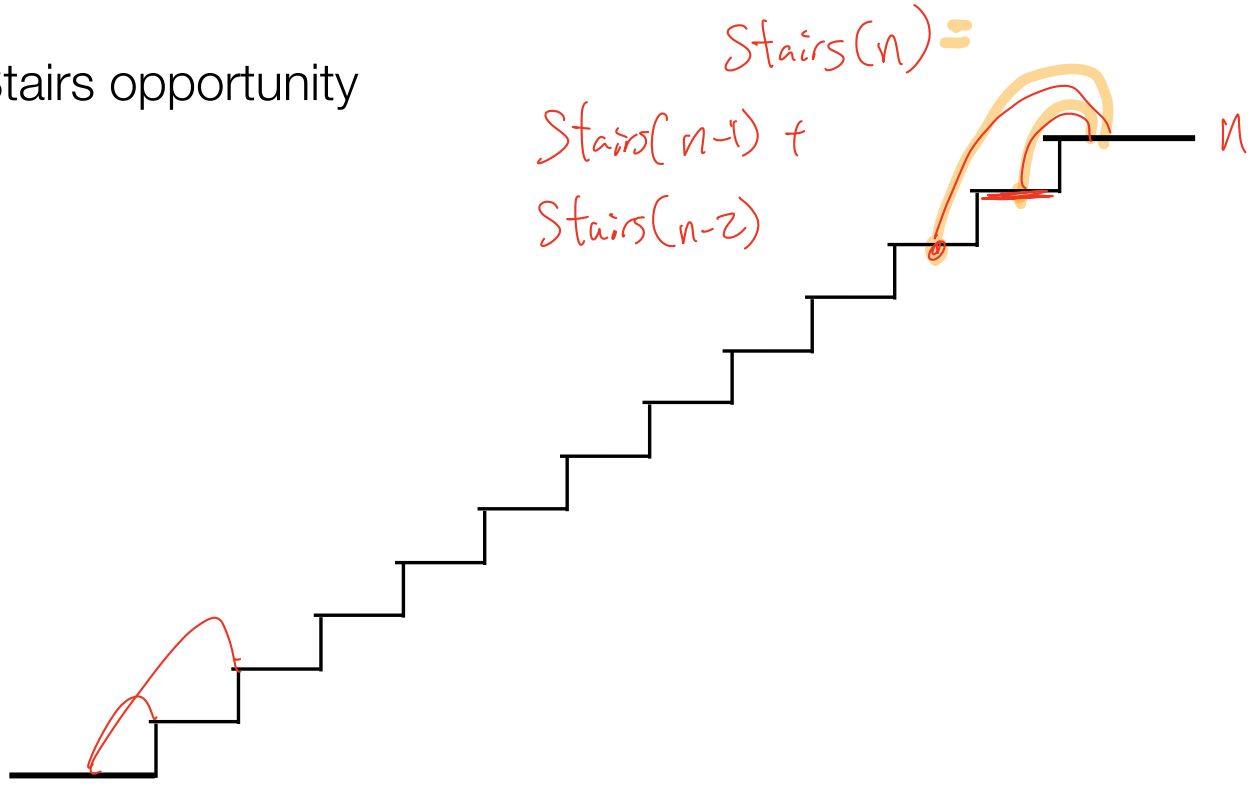
feb 8/10 2022

shelat

We are introducing a new algorithmic technique


West 81st Street, New York, ...
Add a photo

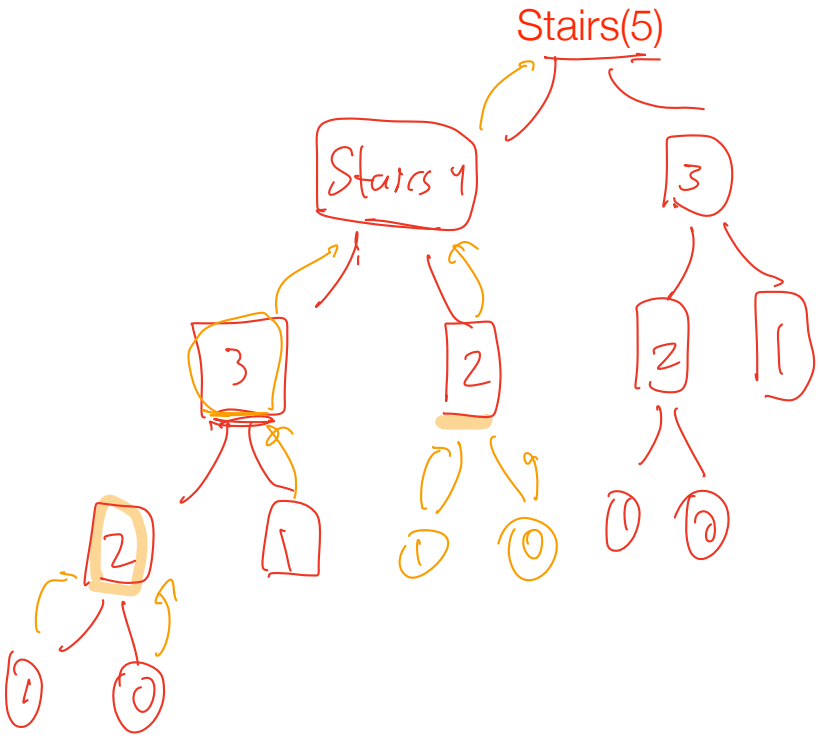
The Stairs opportunity



```
Stairs(n)
    if n<=1 return 1
    return Stairs(n-1) + Stairs(n-2)
```

Stairs(n)

```
if n<=1 return 1
Return Stairs(n-1) + Stairs(n-2)
```



Stairs(n)

```
if n<=1 return 1
Return Stairs(n-1) + Stairs(n-2)
```

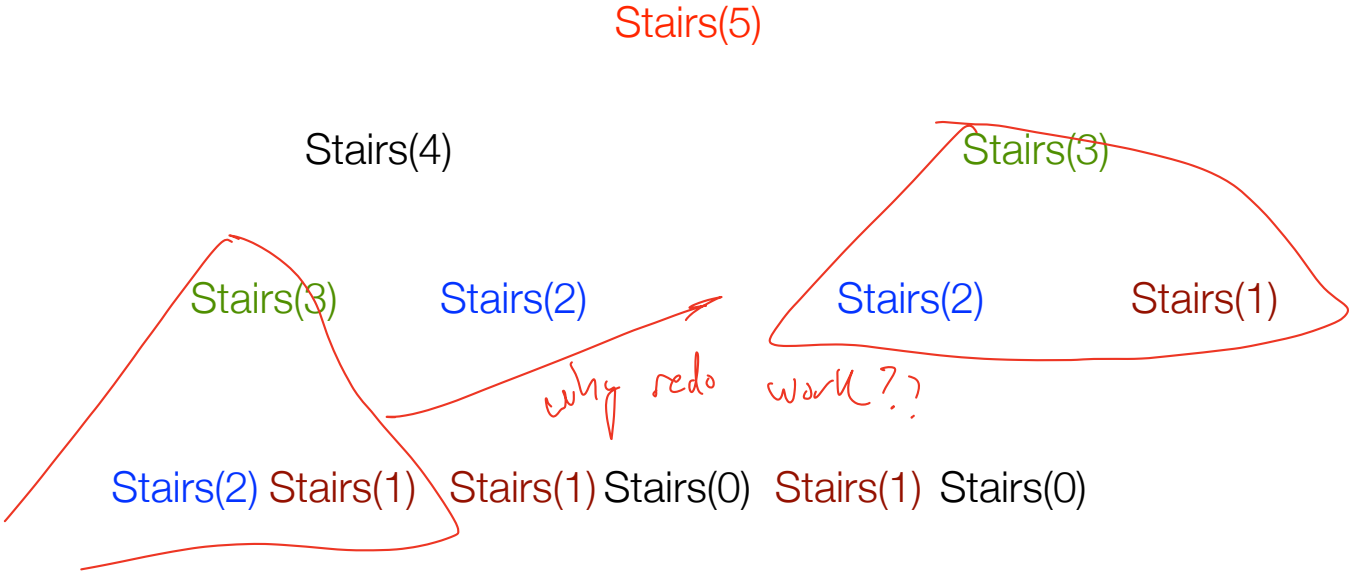
Stairs(5)

Stairs(4)

Stairs(3)

Stairs(n)

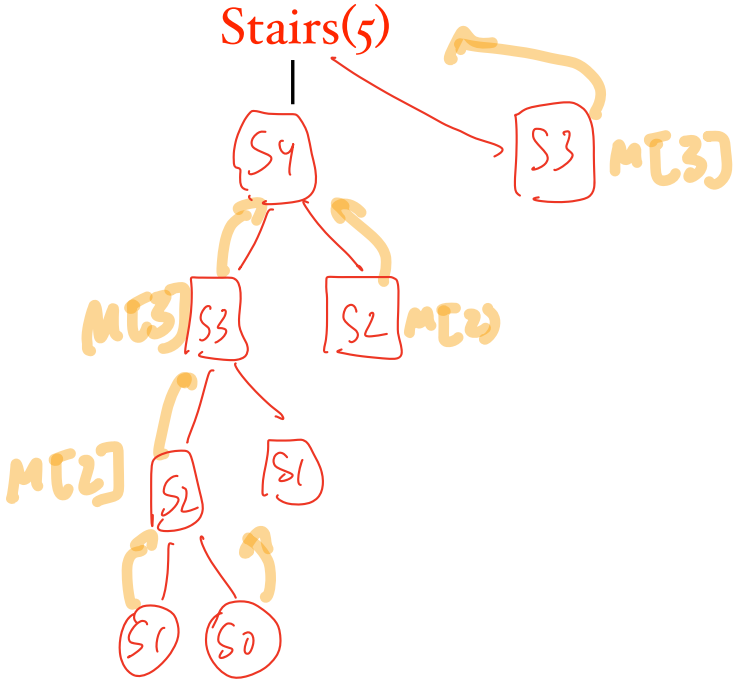
```
if n<=1 return 1
Return Stairs(n-1) + Stairs(n-2)
```



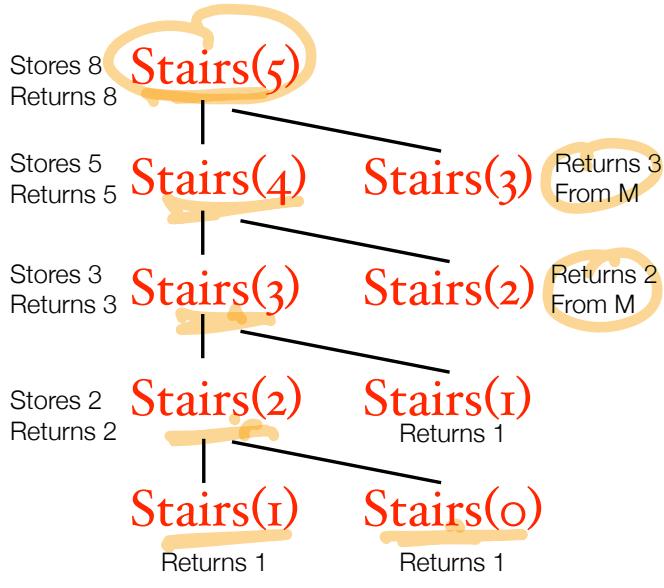
```
initialize memory M

Stairs(n)
```

```
Stairs(n)
    if n<=1 then return 1
    if n is in M, return M[n]
    answer = Stairs(i-1)+ Stairs(i-2)
    M[n] = answer
    return answer
```

store answer



```
Stairs(n)

    if n<=1 then return 1
    if n is in M, return M[n]
    answer = Stairs(i-1)+ Stairs(i-2)
    M[n] = answer
    return answer
```



Stairs(n)

```
stair[0]=1
stair[1]=1
```

Stairs(n)

```
stair[0]=1
stair[1]=1
for i=2 to n
   stair[i] = stair[i-1]+stair[i-2]
return stair[i]
```

For this simple example, you might have started with this structure. But the same pattern applies to more complicated examples.

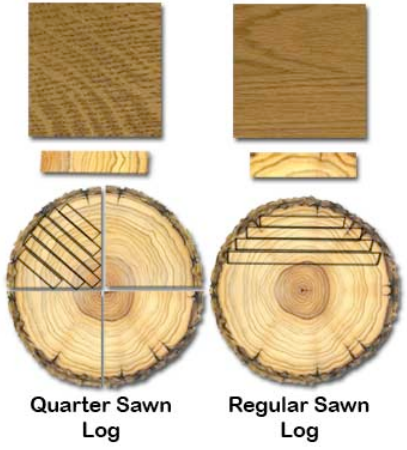
Dynamic Programming

two ideas

recursive structure

memoizing

wood cutting



http://www.amishhandcraftedheirlooms.com/quarter-sawn-oak.htm

http://snlm.files.wordpress.com/2008/08/bill-wakefield-and-carl-fie.gif

Spot price for lumber

1”	2”	3”	4”	5”	6”	7”	8”
1$	4$	9$	20$	. . .			192$

Log cutter dilemna

input to the problem: width n and prices $(p_1, \ldots, p_n)$

goal: how can you maximize the revenue you can get from a tree of width n??

Log cutter dilemna

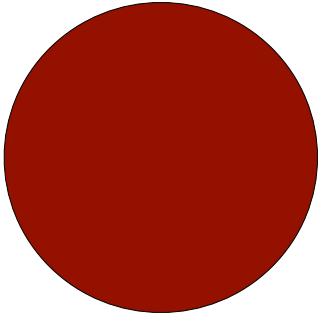
input to the problem: width n and prices $(p_1, \ldots, p_n)$

goal: Output plank widths $i_1, \ldots, i_k$ such that the sum of the plank widths is less than n which maximizes the revenue $\sum_{j=1}^{k} p_{i_j}$

Greedy fails

1”	2”	3”	4”	5”
1$	6$	7$	8$	10$

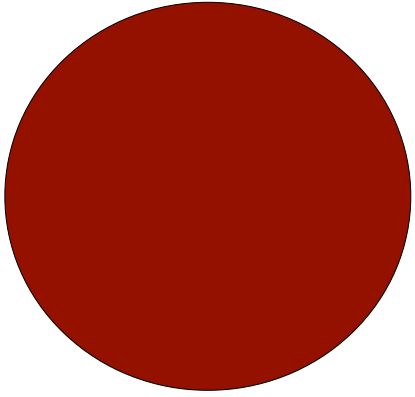
5” log



Greedy “Avg” fails

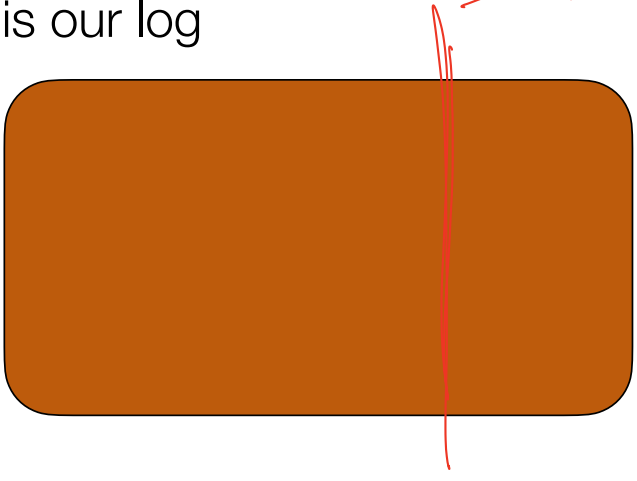
1”	2”	3”	4”	5”	6”
1$	18$	24$	36$	50$	50$

6” log



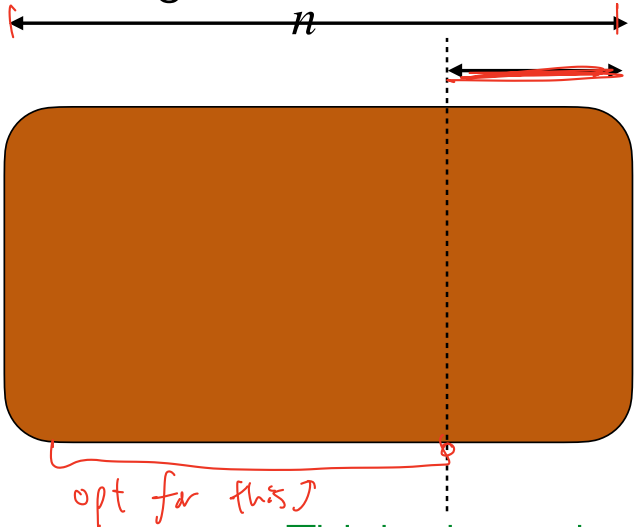
Observation

This is our log



Observation

This is our log



Last cut of length i_ℓ

$opt_n = p_{i_\ell} + opt_{n-i_\ell}$

how many possible cuts could one make??

Think about the very last cut that is made in the optimal solution. From this, we know that the best revenue is the price of this cut plus the best solution for the rest of the log.

Solution equation

$$Opt_n = \max_{i=1\ldots n} \{ p_i + Opt_{n-i} \}$$

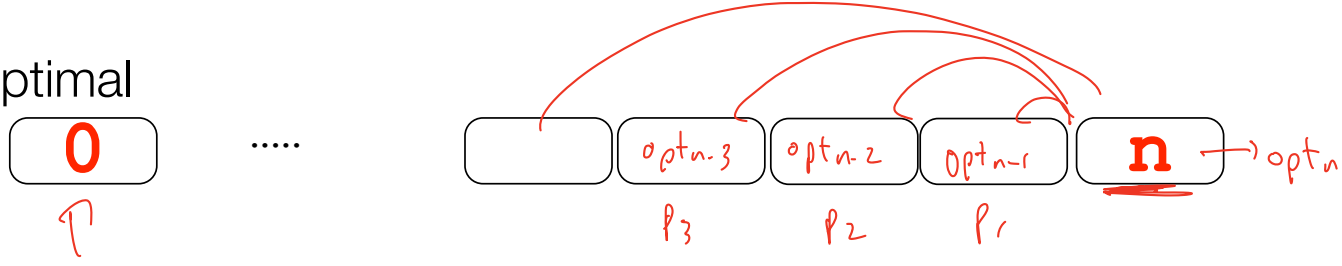
Solution equation

$$opt_n = \max_{1 \ldots n} \left\{ p_i + opt_{n-i} \right\}$$

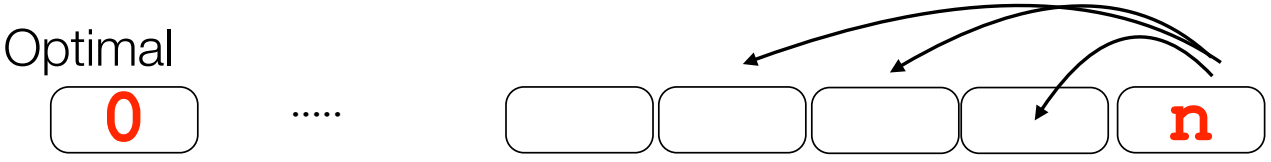
The optimal revenue for a log of n is the max of the price for the last cut, plus the optimal revenue for the remainder of the log.

Approach

Optimal



Approach



opt_n depends on the optimal values for n-1, n-2, …

This suggests to build the array from 0 to n.

BestLogs($n, (p_1, \ldots, p_n)$)

```
    if n<=0 return 0
```

BestLogs($n, (p_1, \ldots, p_n)$)

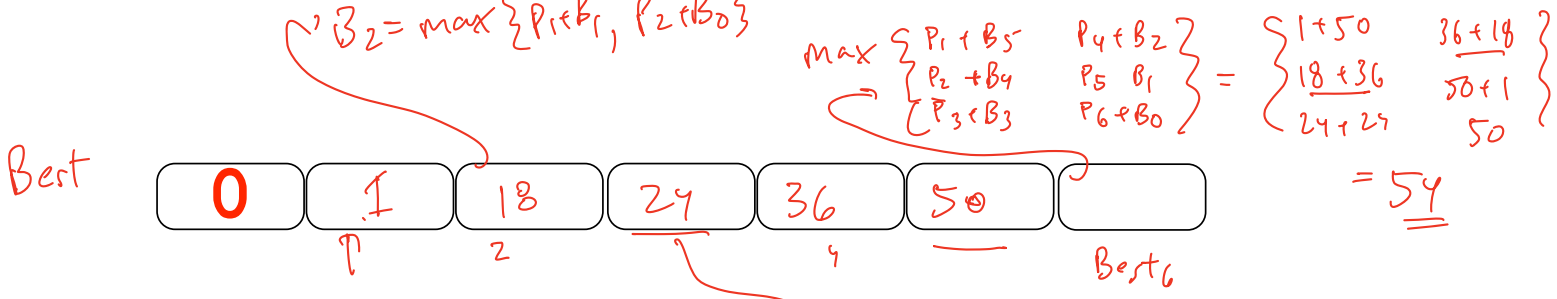
```
if n<=0 return 0
for i=1 to n
 Best[i] = max_{k=1...i} {p_k + Best[i − k]}

return Best[n]
```

Example

1"	2"	3"	4"	5"	6"
1$	18$	24$	36$	50$	50$

BestLogs($n, (p_1, \ldots, p_n)$)

```
if n<=0 return 0
for i=1 to n
  Best[i] = max_{k=1...i} {p_k + Best[i − k]}
return Best[n]
```

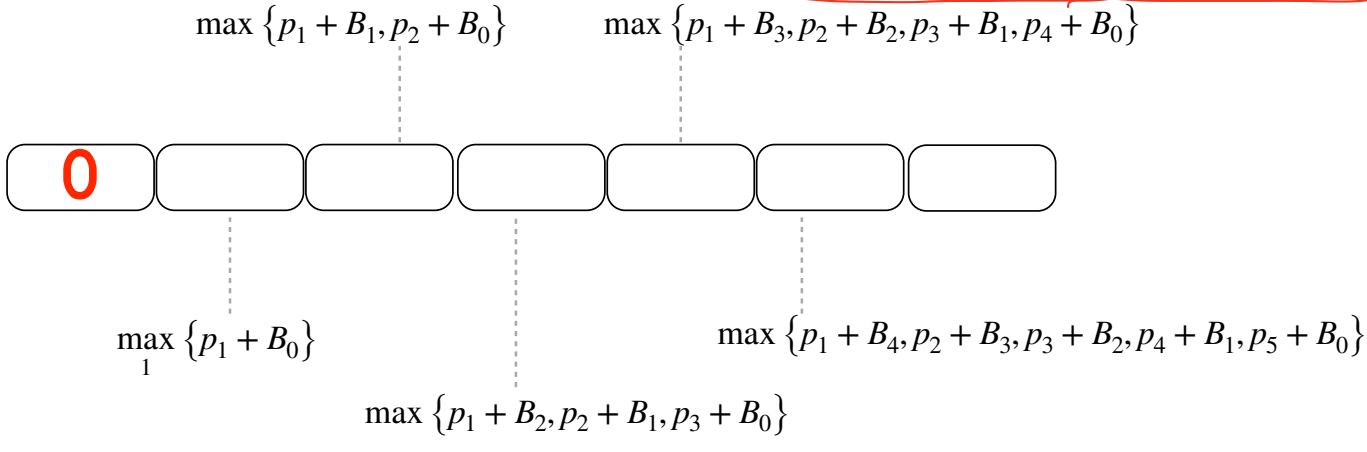


Example

1"	2"	3"	4"	5"	6"
1$	18$	24$	36$	50$	50$

BestLogs($n, (p_1, \ldots, p_n)$)

```
if n<=0 return 0
for i=1 to n
 Best[i] = max_{k=1...i} {p_k + Best[i − k]}
return Best[n]
```



The actual cuts?

The previous algorithm just returns the optimal revenue.
How can you find the optimal cuts?

BestLogs($n, (p_1, \ldots, p_n)$)

```
if n<=0 return 0
for i=1 to n
  Best[i] = max_{k=1...i} {p_k + Best[i − k]}
  choice[i] = k*
return Best[n]
```

loop of n

loop of i.

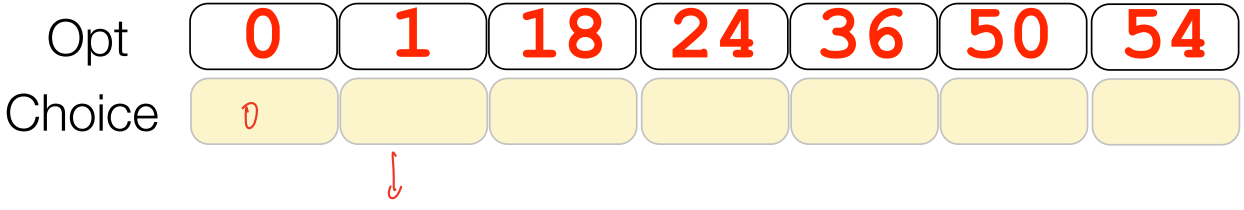
is the index that achieves the max in the Best equation

Running time:

$\Theta(n^2)$

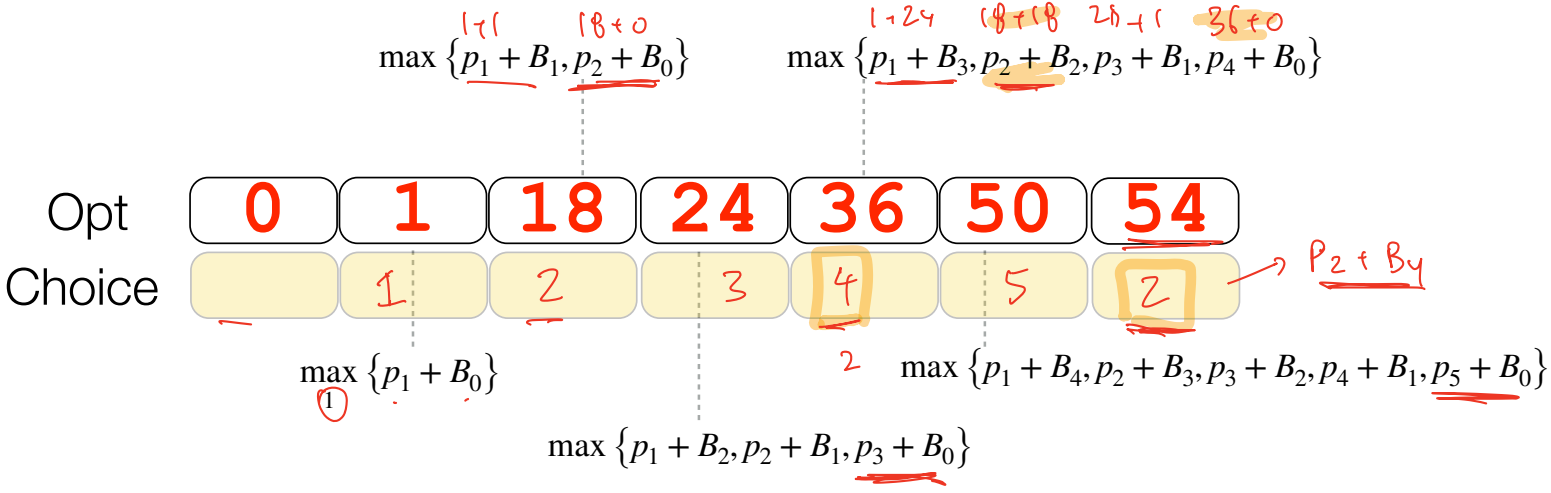
Example

1"	2"	3"	4"	5"	6"
1$	18$	24$	36$	50$	50$



Example

1"	2"	3"	4"	5"	6"
1$	18$	24$	36$	50$	50$



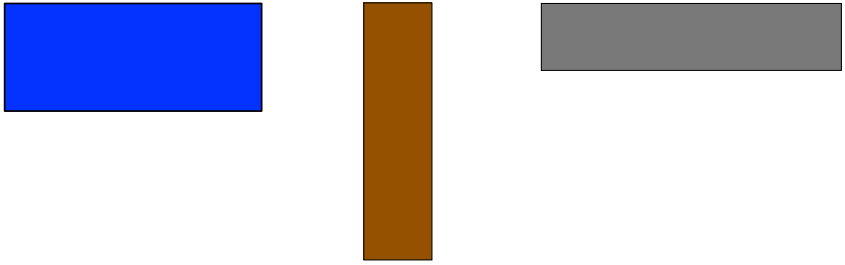
Main ideas

Identify the recursive structure of the problem with an equation.

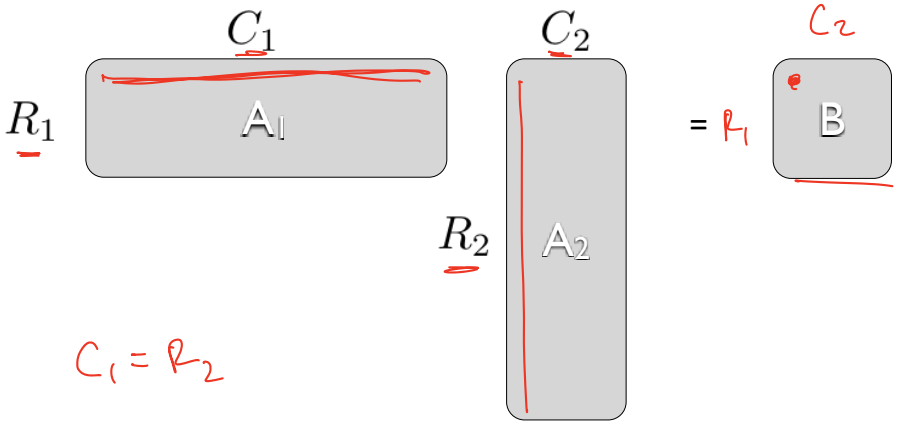
Design a way to compute this equation through an iterative loop that follows a good order.

Use another variable to record optimal parameters at each step to reconstruct a solution.

Matrix



Dynamic programming in 2 dimensions!

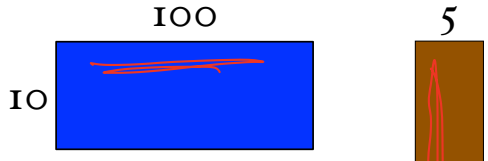


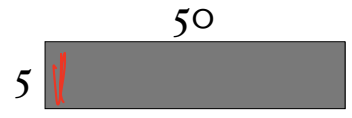
$C_1 = R_2$

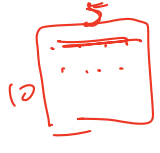
$$A_1 \cdot A_2 \cdot A_3$$

$$(A_1 \cdot A_2) \cdot A_3 \qquad A_1 \cdot (A_2 \cdot A_3)$$

$$(A_1 \cdot A_2) \cdot A_3$$







$5 \cdot 10 \cdot 100$

5000

50 entries

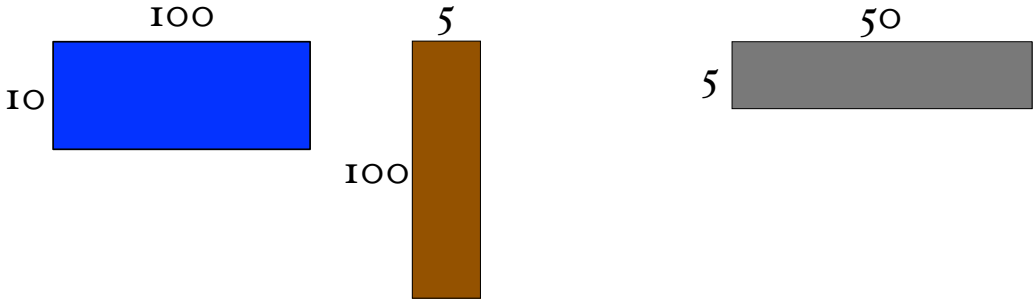
total work

$10 \cdot 50 \cdot 5$

2500

Total 7500

$$(A_1 \cdot A_2) \cdot A_3$$

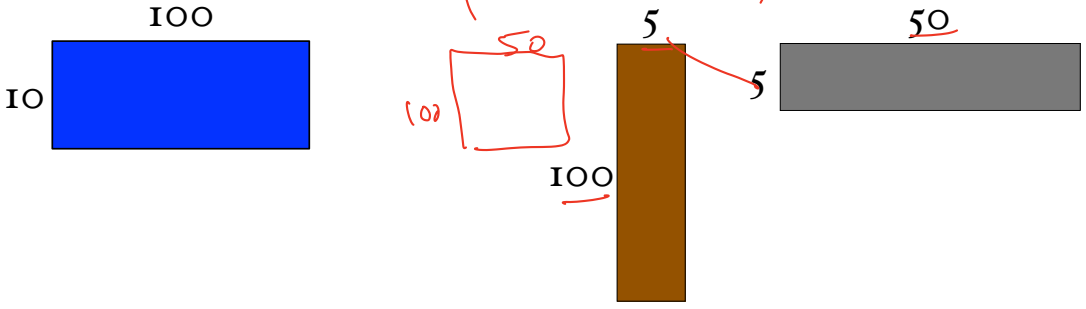


$$10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50$$

operations

7500

$$A_1 \cdot (A_2 \cdot A_3)$$



$10 \cdot 100 \cdot 50 =$

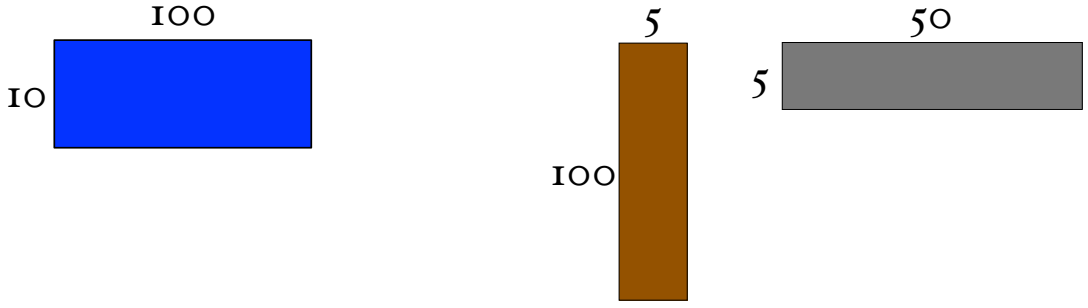
$50{,}000$

$100 \cdot 50 \cdot 5 =$

$25{,}000$

overall: $75{,}000$ operations !!!

$$A_1 \cdot A_2 \cdot A_3$$



$$100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50$$

operations

order matters

(for efficiency)

Key Question: what is the best order in which to multiply the matrices?

how do we solve it?

identify smaller instances of the problem

devise method to combine solutions

small # of different subproblems

solved them in the right order

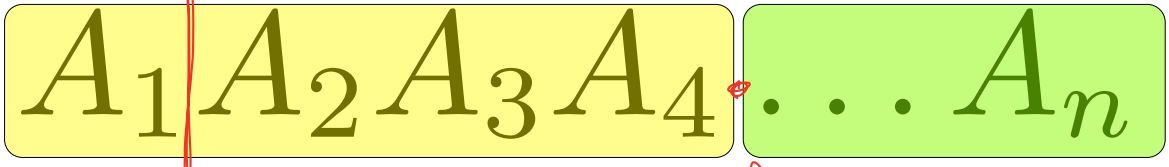
optimal way to compute

$$A_1 A_2 A_3 A_4 \cdot A_5 \cdot \ldots \cdot A_n$$

Suppose this mult was the last step in the ~~optional~~ optimal order for computing the product.

$Opt_{1,n} = Opt_{1,4} + Opt_{5,n} +$

optimal way to compute



Suppose this mult was the last step in the optional order for computing the product.

$$B(1,n) = B(1,4) + B(5,n) + ?$$

$r_1 \cdot c_4 \cdot c_n$

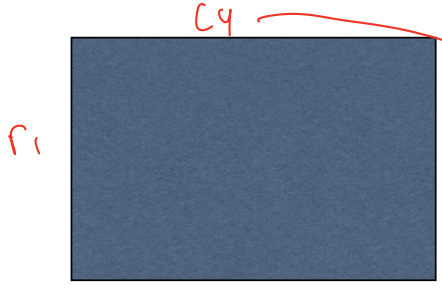
cost of multiplying these last two??

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

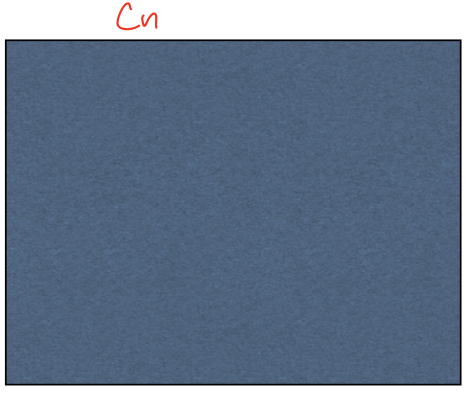
$r_1 \cdot c_4 \cdot c_n$

Size of $A_1 A_2 A_3 A_4$



=

Size of $A_5 \ldots A_n$



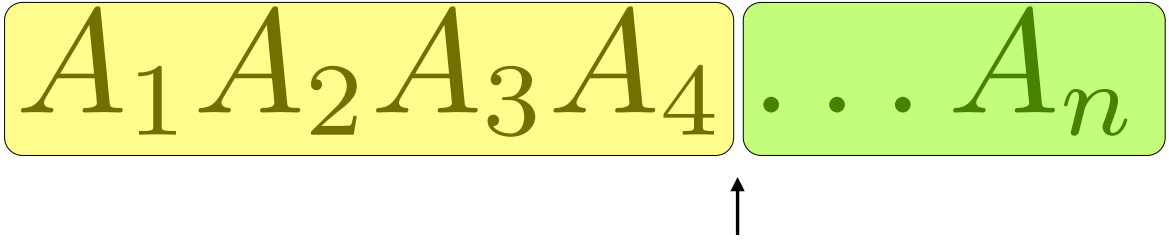
$A_1A_2A_3A_4$

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

Cost of multiplying $A_1 A_2 A_3 A_4$ and $A_5 \ldots A_n$

optimal way to compute



optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

Suppose this mult was the last step in the optional order for computing the product.

$$B(1,n) = B(1,4) + B(5,n) + ?$$

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

B[1,n] =

min

B(1,1) + B(2,n) + $r_1 \cdot c_1 \cdot c_n$

B(1,2) + B(3,n) + $r_1 \cdot c_2 \cdot c_n$

+ ... +

B(1,i) + B(i+1,n) + $r_1 \cdot c_i \cdot c_n$

B(1,n-1) + B(n,n) + $r_1 \cdot c_{n-1} \cdot c_n$

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

B[1,n]

B[1,1]
B[2,n]

$R_1 C_1 C_n$

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

B[1,n]

min

B[1,1]	B[1,2]	...	B[1,n-2]	B[1,n-1]
B[2,n]	B[3,n]	...	B[n-1,n]	B[n,n]
$R_1 C_1 C_n$	$R_1 C_2 C_n$		$R_1 C_{n-2} C_n$	$R_1 C_{n-1} C_n$

$$B(i,i) = 1$$

$$B(1,n) = \min \begin{cases} \\ \\ \\ \end{cases}$$

$$B(i,i) = 1$$

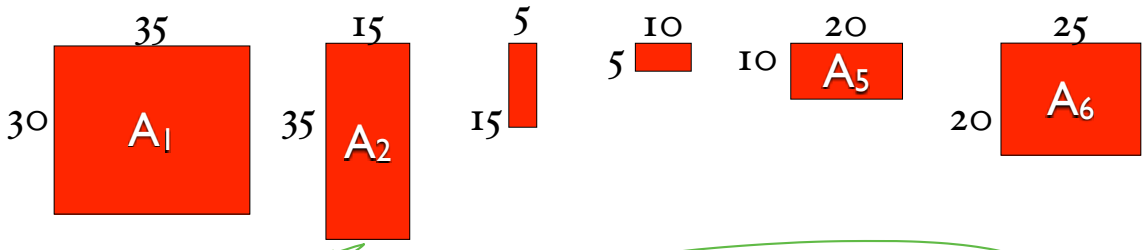
$$B(1,n) = \min \begin{cases} B(1,1) + B(2,n) + r_1 c_1 c_n \\ B(1,2) + B(3,n) + r_1 c_2 c_n \\ \vdots \\ B(1,n-1) + B(n,n) + r_1 c_{n-1} c_n \end{cases}$$

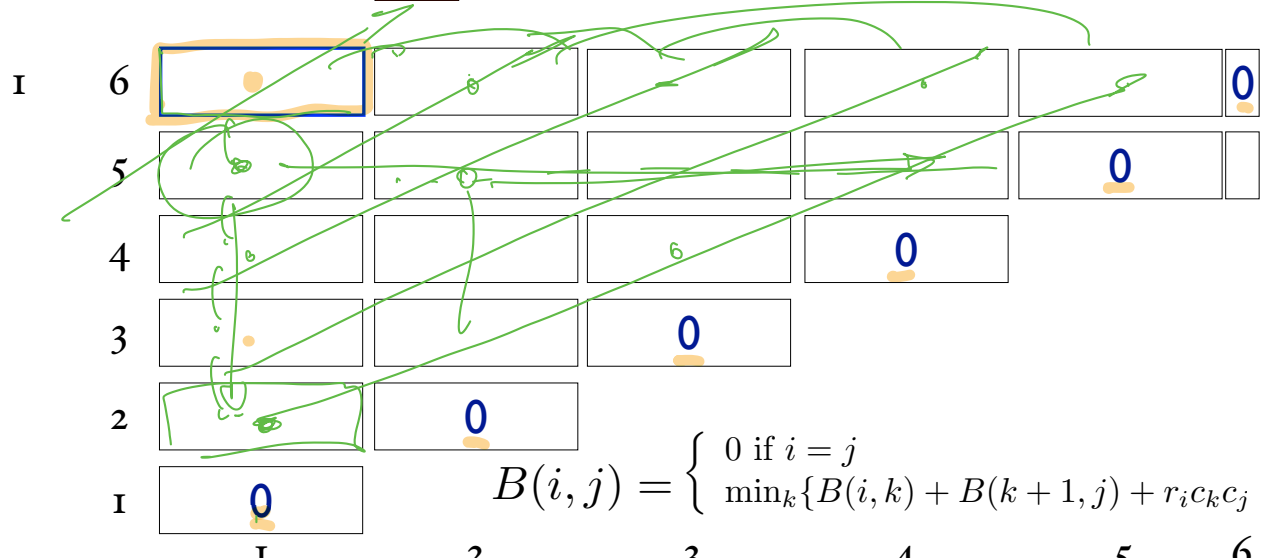
$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

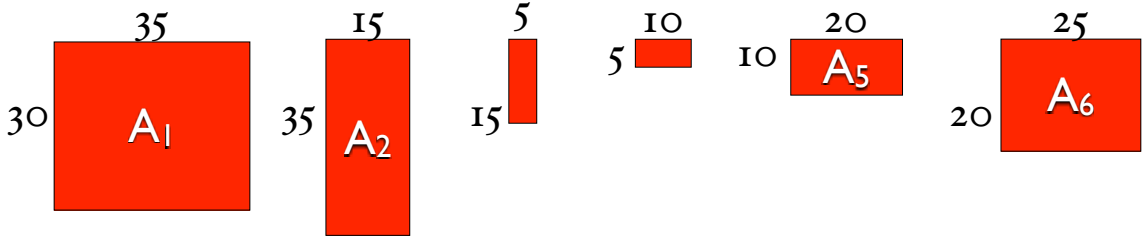
which order to solve?

B(1,6)



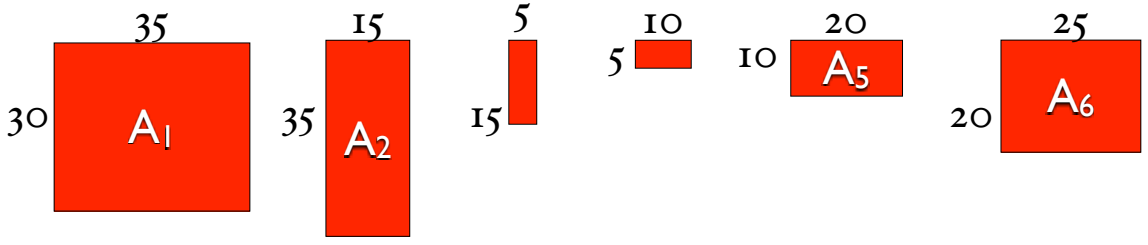


$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$



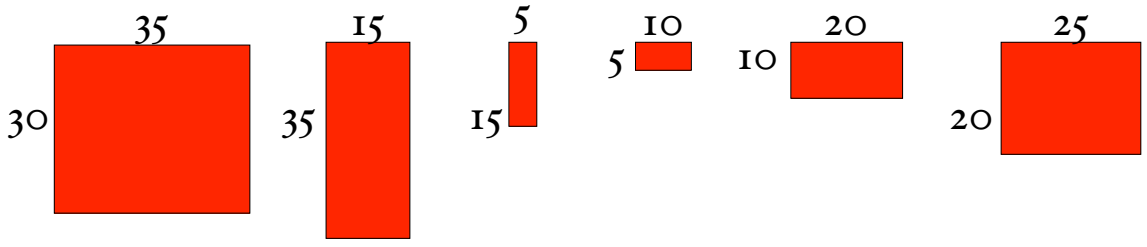
1

$$B(1,2) = \min \left\{ \begin{array}{lll} B(1,1) + B(2,2) + r_1 \cdot c_1 \cdot c_2 \\ 0 \qquad\quad 0 \qquad\quad 30 \cdot 35 \cdot 15 \end{array} \right\}$$



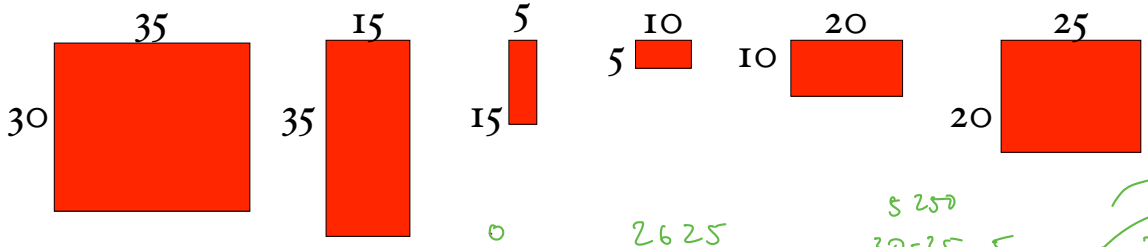
1

$$B(1,2) = \min\left\{B(1,1) + B(2,2) + r_1 c_1 c_2\right\}$$



6					10*20*25 = 5000	0
5				5*10*20 = 1000	0	
4			15*5*10 = 750	0		
3		35*15*5 = 2625	0			
2	30*35*15 = 15750	0				
1	0					
	1	2	3	4	5	6

$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$



$$B(1,3) = \min \{$$

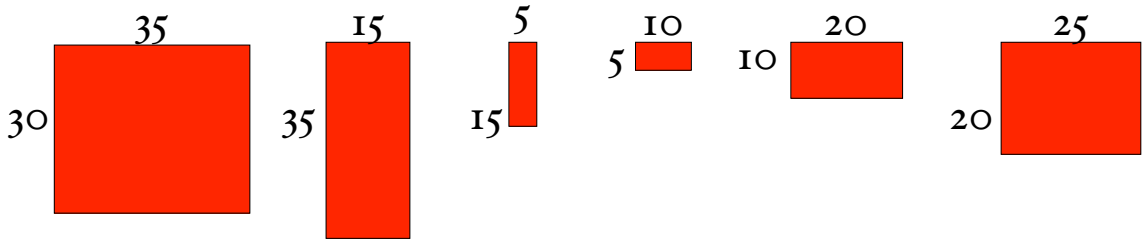
$B(1,1) + B(2,3) + r_1 \cdot c_2 \cdot c_3$ → 0 + 2625 + 5250 = 7875

$B(1,2) + B(3,3) + r_1 \cdot c_2 \cdot c_3$ → 15750 + 0 + 30·15·5

3		35*15*5 = 2625	0
2	30*35*15 = 15750	0	
1	0		
	1	2	3

$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

4 5 6

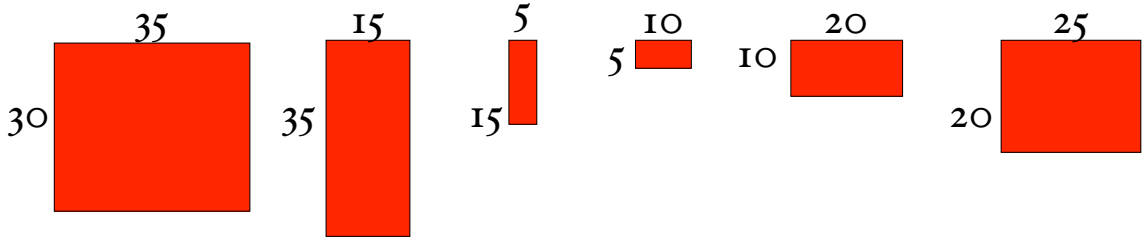


B(1,6)

1	6		10500	5375	3500	10*20*25 = 5000	0
	5	11875	7125	2500	5*10*20 = 1000	0
	4	9375	4375	15*5*10 = 750	0	
	3	7875	35*15*5 = 2625	0		
	2	30*35*15 = 15750	0			
	1	0				
		1	2	3	4	5	6

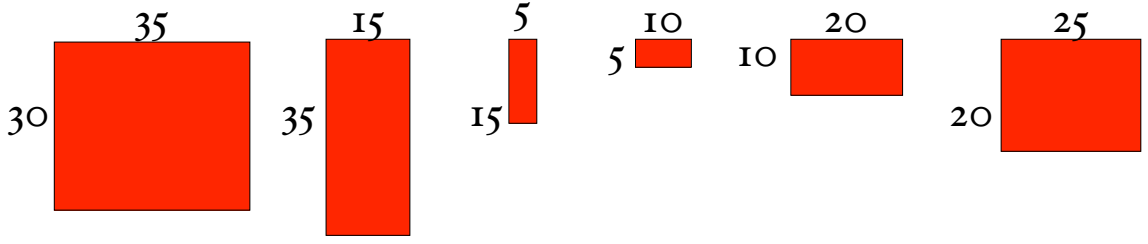
$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

i, j-1



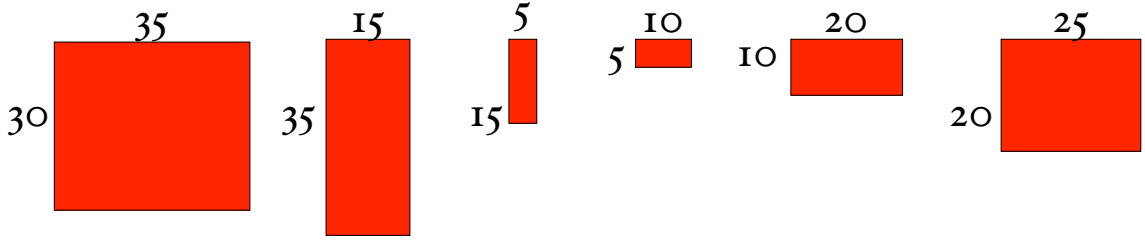
1 6

$$C(1,6) = \min \begin{cases} k=1 & C(1,1) + C(2,6) + r_1 c_1 c_6 \\ k=2 & C(1,2) + C(3,6) + r_1 c_2 c_6 \\ k=3 & C(1,3) + C(4,6) + r_1 c_3 c_6 \\ k=4 & C(1,4) + C(5,6) + r_1 c_4 c_6 \\ k=5 & C(1,5) + C(6,6) + r_1 c_5 c_6 \end{cases}$$



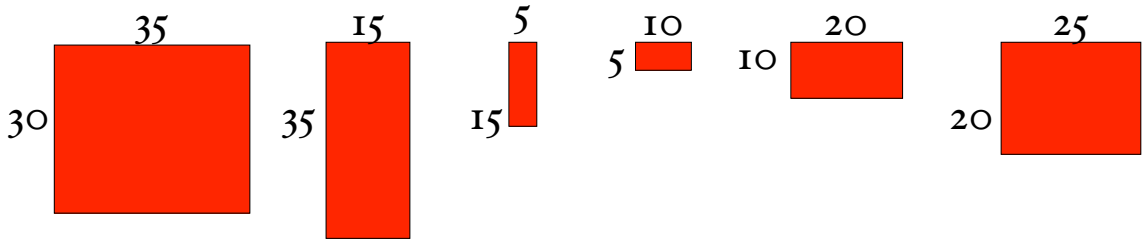
1 6

$$C(1,6) = \min \begin{cases} k=1 & 0 + 10500 + 30 \cdot 35 \cdot 25 \\ k=2 & 15750 + 5375 + 30 \cdot 15 \cdot 25 \\ k=3 & 7875 + 3500 + 30 \cdot 5 \cdot 25 \\ k=4 & 9375 + 5000 + 30 \cdot 10 \cdot 25 \\ k=5 & 11875 + 0 + 30 \cdot 20 \cdot 25 \end{cases}$$

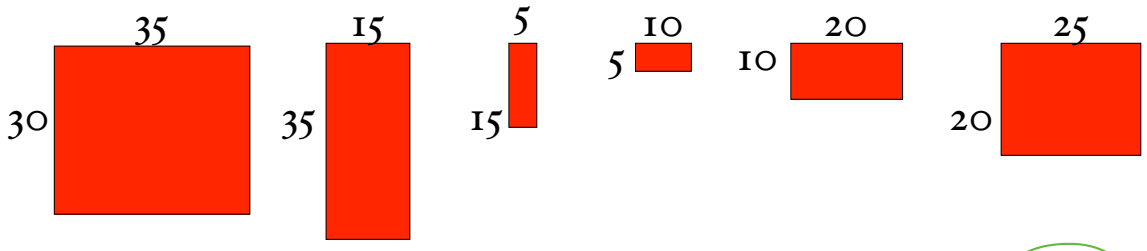


1 6

$$C(1,6) = \min \begin{cases} k=1 & 0 + 10500 + 26250 \\ k=2 & 15750 + 5375 + 11250 \\ k=3 & 7875 + 3500 + 3750 \\ k=4 & 9375 + 5000 + 7500 \\ k=5 & 11875 + 0 + 15000 \end{cases}$$



1	6	15125	10500	5375	3500	10*20*25 = 5000	0
	5	11875	7125	2500	5*10*20 = 1000	0	
	4	9375	4375	15*5*10 = 750	0		
	3	7875	35*15*5 = 2625	0			
	2	30*35*15 = 15750	0				
	1	0					
		1	2	3	4	5	6



1	6	15125 3	10500	5375	3500	10*20*25 = 5000	0
	5	11875	7125	2500	5*10*20 = 1000	0	
	4	9375	4375	15*5*10 = 750	0		
	3	7875	35*15*5 = 2625	0			
	2	30*35*15 = 15750	0				
	1	0					
		1	2	3	4	5	6

matrix-chain-mult(p)

initialize array m[x,y] to zero

matrix-chain-mult(p)

initialize array m[x,y] to zero

starting at diagonal, working towards upper-left

compute m[i,j] according to

$$\begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

choice i,j = k*

running time?

initialize array m[x,y] to zero

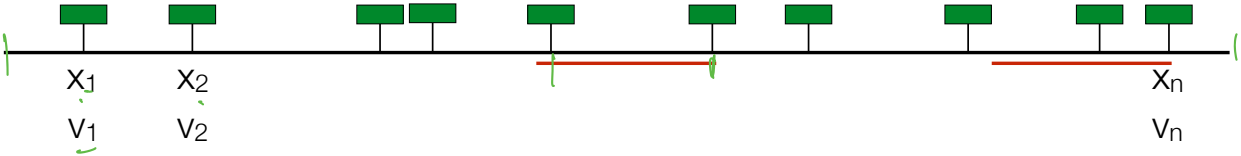
starting at diagonal, working towards upper-left

compute m[i,j] according to

$$\begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

Billboard problem

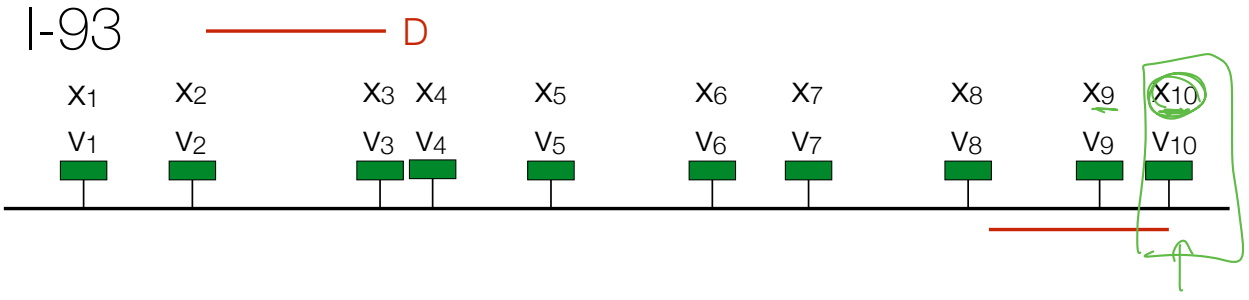
I-93



D

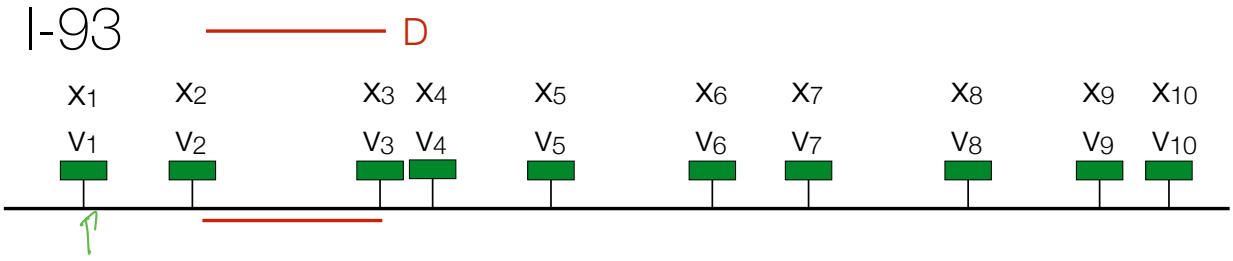
distance parameter

Cannot place ads that are closer than D miles apart



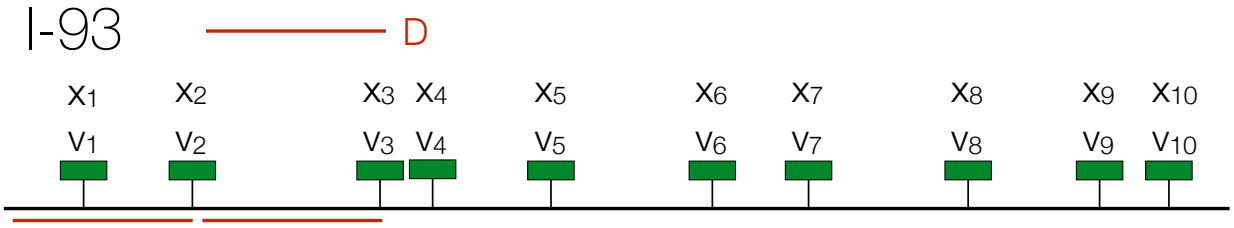
Input is $((x_1, \ldots, x_n)(v_1, \ldots, v_n), D)$

$\text{Best}_n = \max \begin{cases} v_{10} + \text{Best}[\quad] \\ \text{Best}_{n-1} \end{cases}$



$Best_1 =$

$Best_3 =$



$Best_1$ =

$Best_2$ =

$Best_3$ =

Billboard Problem

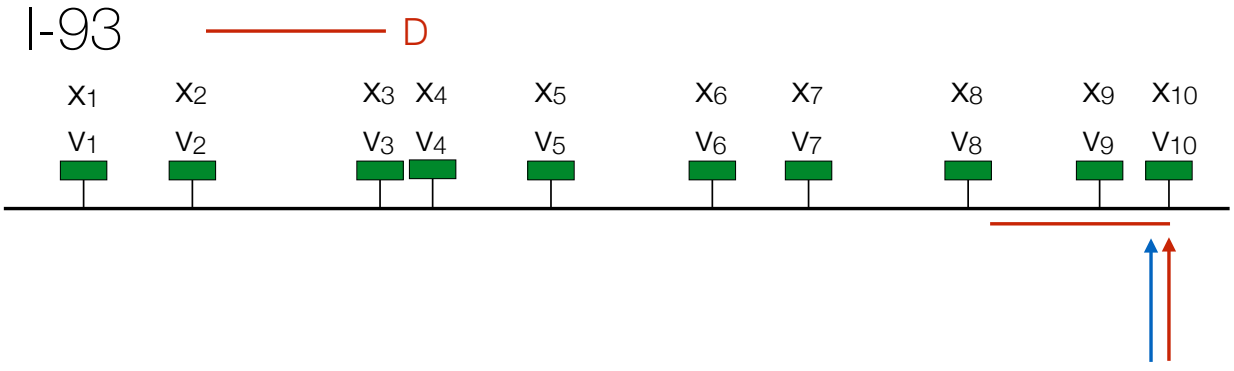
$$\text{BEST}_j = \max \begin{cases} \text{BEST}_{j-1} \\ v_j + \text{BEST}_{cl(j)} \end{cases}$$

```
best[0] = 0
for i=1 to n




return best[n]
```

Billboard Problem

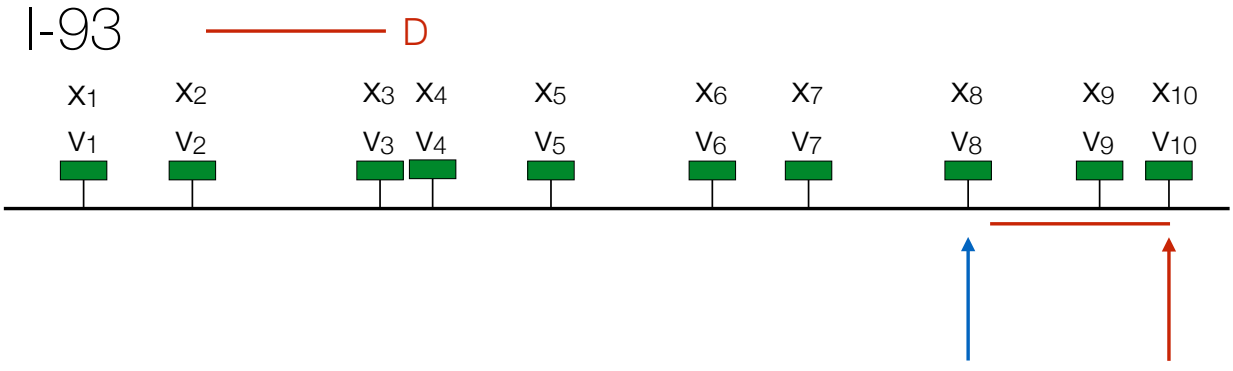
$$\text{BEST}_j = \max \begin{cases} \text{BEST}_{j-1} \\ v_j + \text{BEST}_{cl(j)} \end{cases}$$

```
best[0] = 0
for i=1 to n
    cl = i-1
    while( (x[i]-x[cl])< D  && cl>0) cl=cl-1
    best[i] = max(best[i-1], vj+best[cl])
return best[n]
```



Pre-process to find every board's buddy.

right = n, left = n

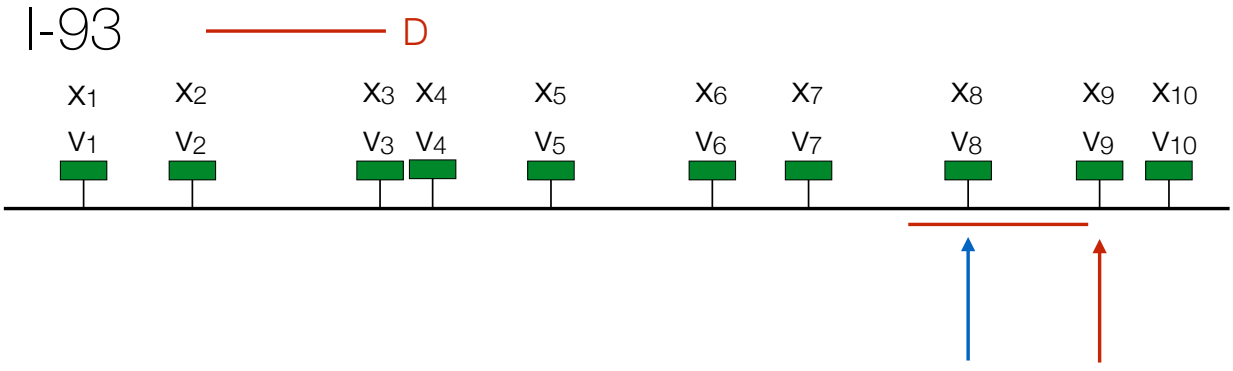


Pre-process to find every board's buddy.

right = n, left = n

move left until dist(x[right], x[left]) > D

buddy[right] = left



Pre-process to find every board's buddy.

b[10]=8

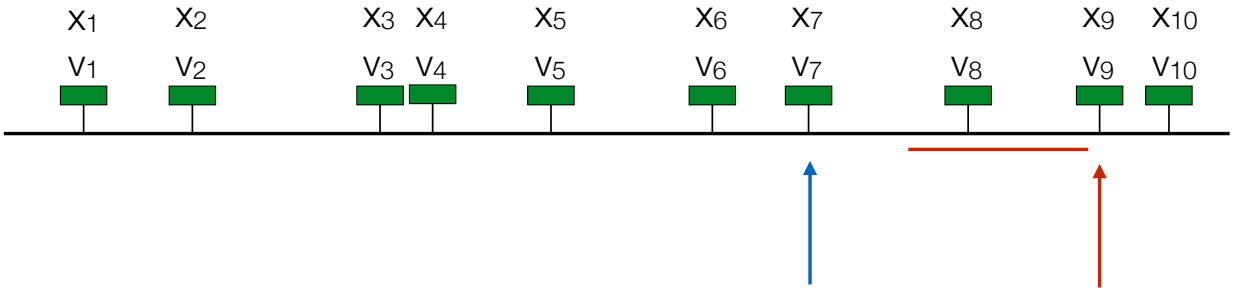
right = n, left = n

move left until dist(x[right], x[left]) > D

buddy[right] = left

move right to right

I-93 D



Pre-process to find every board's buddy.

b[10]=8

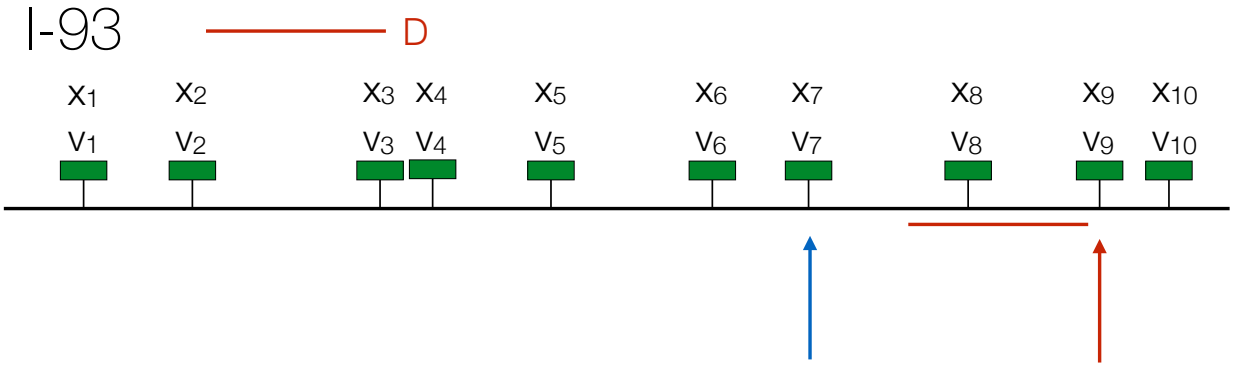
right = n, left = n

while right and left are valid

move left until dist(x[right], x[left]) > D

buddy[right] = left

move right to right



Pre-process to find every board's buddy.

b[10]=8

right = n, left = n

while right and left are valid

move left until dist(x[right], x[left]) > D

buddy[right] = left

move right to right

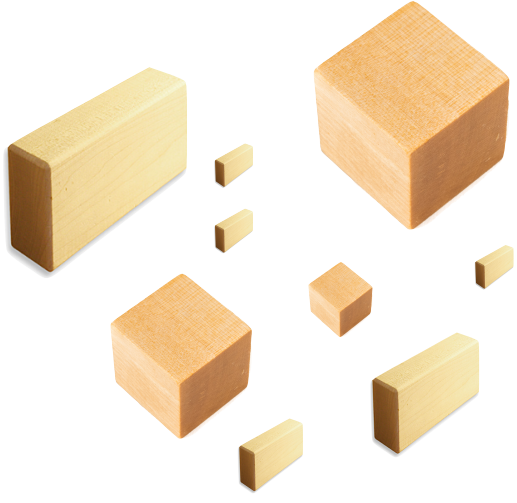
handle any leftover right

Better Billboard

$$\text{BEST}_j = \max \begin{cases} \text{BEST}_{j-1} \\ v_j + \text{BEST}_{cl(j)} \end{cases}$$

```
<Preprocess buddies>
best[0] = 0
for i=1 to n
   cl = i-1
   while( (x[i]-x[cl])< D  && cl>0) cl=cl-1
   best[i] = max(best[i-1], v[j]+best[buddy[i]])
return best[n]
```

Knapsack



Sack has Capacity W

W

w_1 v_1

w_2 v_2

w_3 v_3

Each item
has a weight w_i
and a value v_i

Goal is to select
a set of items that
"**fit**" into the Knapsack
and have the **greatest**
value.

Define a quantity that captures the optimal solution:

Best({1,…,n}, C) :

Consider the very first item. Is it part of the max solution?

Define a quantity that captures the optimal solution:

Best({1,…,n}, C) : max value obtainable from items {1…n} that fit in sack of size C

Consider the very first item. Is it part of the max solution?

Define a quantity that captures the optimal solution:

Best({1,…,n}, C) : max value obtainable from items {1…n} that fit in sack of size C

Consider the very first item. Is it part of the max solution?

$$B(1\ldots n, C) = \max \begin{cases} B(2\ldots n, C) & \text{if not included} \\ v_1 + B(2\ldots n-1, C-w_1) & \text{if in} \end{cases}$$

Recursive structure

Either the best solution doesn't include item i

B({i…n}, C) = max

Or, it includes item i and the best solution for the remaining space, $C-w_i$

Recursive structure

Either the best solution doesn't include item i

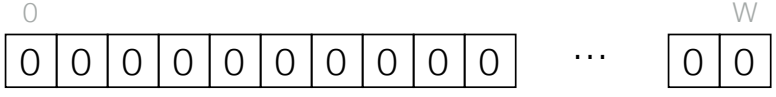
$$B(\{i\ldots n\}, C) = \max \begin{cases} B(\{i+1\ldots n\}, C) \\ v_i + B(\{i+1\ldots n\}, C - w_i) \end{cases}$$

Or, it includes item i and the best solution for the remaining space, $C-w_i$

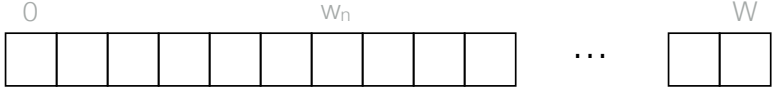
Pick an order

Start from the last item

B({ }, 0…W)

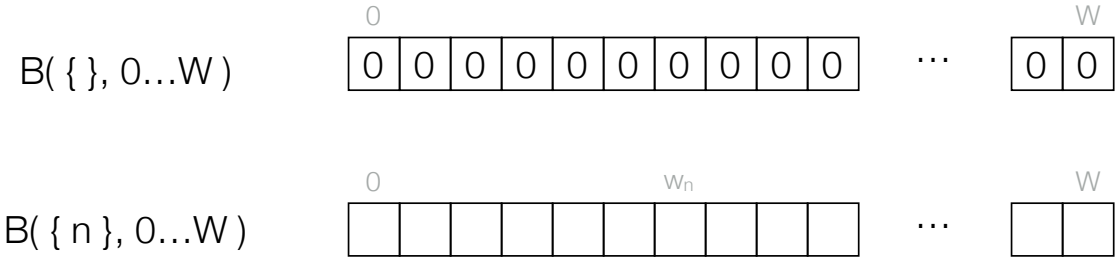


B({ n }, 0…W)



Pick an order

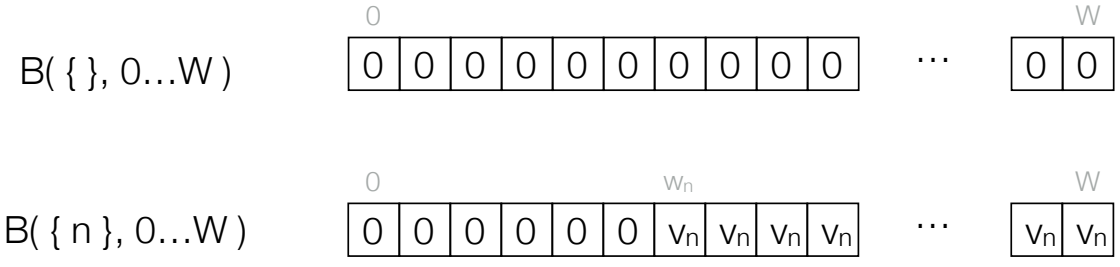
Start from the last item



$$B(\{n\}, C) = \max \begin{cases} B(\{\}, C) \\ v_n + B(\{\}, C - w_n) \end{cases}$$

Pick an order

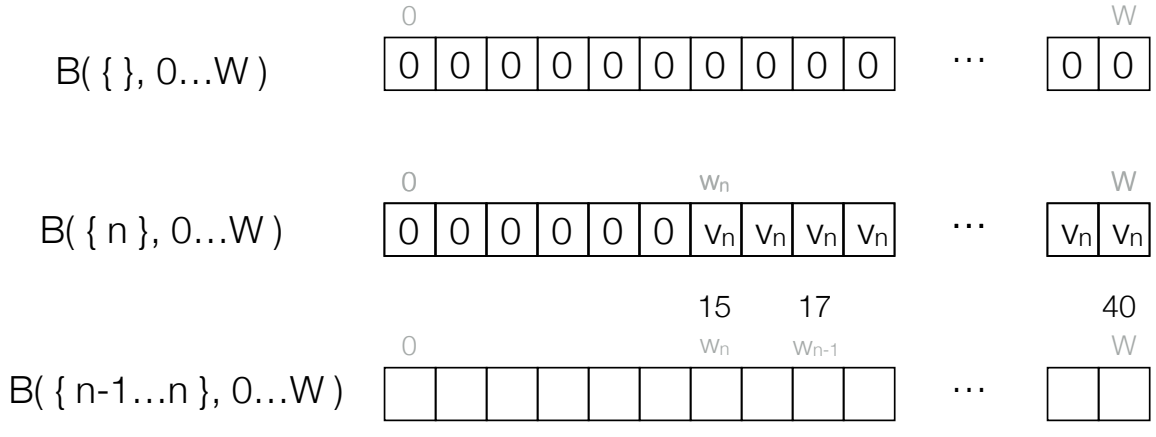
Start from the last item



$$B(\{n\}, C) = \max \begin{cases} B(\{\}, C) \\ v_n + B(\{\}, C - w_n) \end{cases}$$

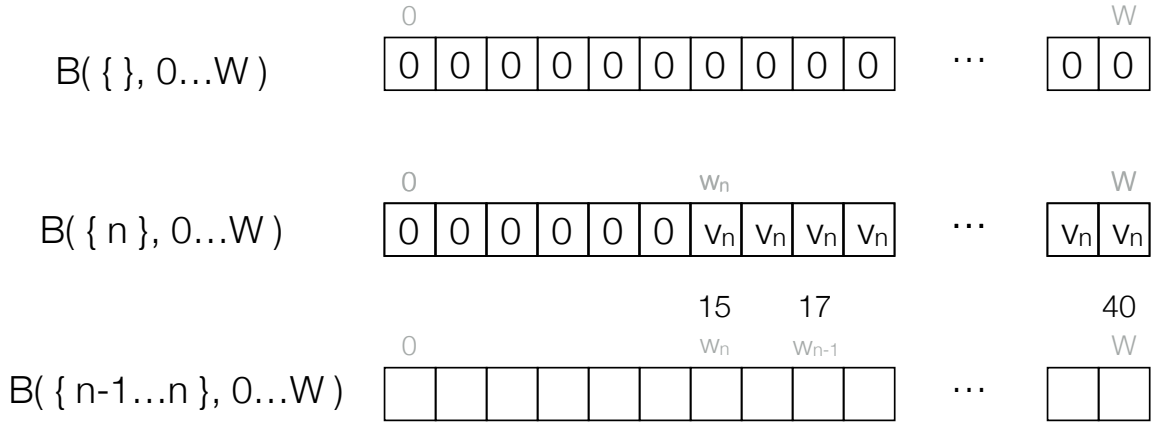
Pick an order

Start from the last item



Pick an order

Start from the last item



$$B(\{i \ldots n\}, C) = \max \begin{cases} B(\{i+1 \ldots n\}, C) \\ v_i + B(\{i+1 \ldots n\}, C - w_i) \end{cases}$$

Knapsack($\{w_i, v_i\}_n$, W)

Initialize B({n-1}, 0…W) = 0

for i from n to 1

for j from 0 to W

B(i, j) = max

$$\begin{cases} B(i+1, j) \\ v_i + B(i+1, j - w_i) \end{cases}$$

as long as $j > w_i$ because otherwise, this term is negative

Return B(1,W)

Knapsack($\{w_i, v_i\}_n$, W)

Initialize B({n-1}, 0…W) = 0

for i from n to 1

    for j from 0 to W

        B(i, j) = B(i+1, j)

        if j > wi and B(i+1, j-wi)+ vi > S(i, j)
            B(i,j) = B(i+1, j-wi)+ vi

Return B(1,W)

How can we determine WHICH items are selected?

Knapsack($\{w_i, v_i\}_n$, W)

```
Initialize B( {n-1}, 0…W) = 0

for i from n to 1

        for j from 0 to W

            B( i, j ) = B( i+1, j)
            pick( i, j ) = false
            if j > wi and B( i+1, j-wi )+ vi > B( i, j)
                B(i,j) = B( i+1, j-wi )+ vi
                  pick( i, j ) = true

//Backtrack to find solution
cap = W, sol = {}
for i from 1 to n

        if picked(i,cap) = true { sol = sol + {i};  cap = cap - wi; }
```

PROBLEM: REDUCE IMAGE WIDTH

scaling: distortion

deleting column: distortion

delete the most invisible seam

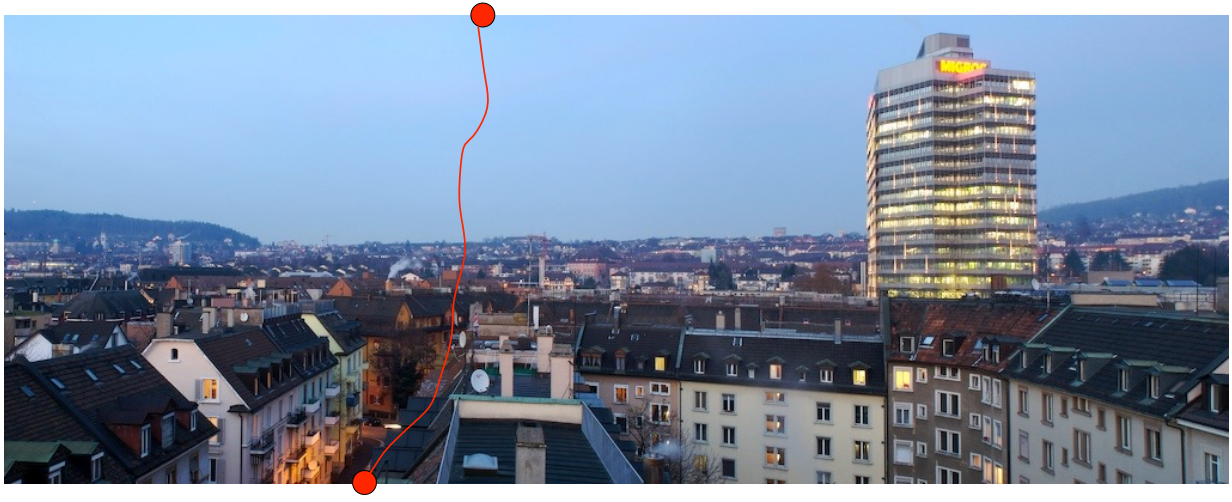
http://www.youtube.com/watch?v=qadw0BRKeMk

http://www.youtube.com/watch?v=qadw0BRKeMk

DEMO?

http://rsizr.com/

WHICH SEAM TO DELETE?



ENERGY OF AN IMAGE

$$e(\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right|$$

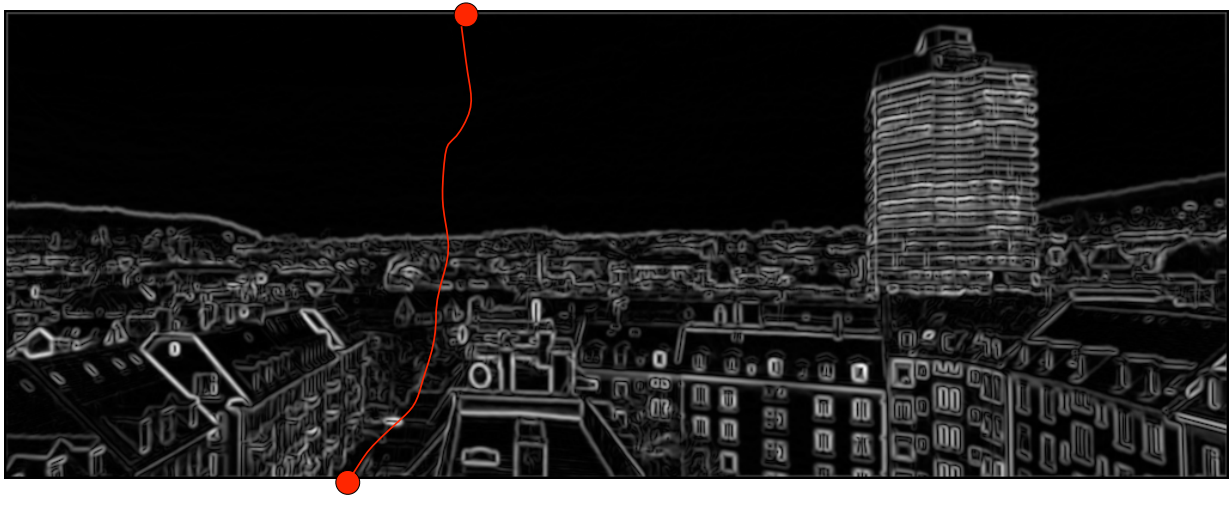
“magnitude of gradient at a pixel”

$$\frac{\partial}{\partial x} I_{x,y} = I_{x-1,y} - I_{x+1,y}$$

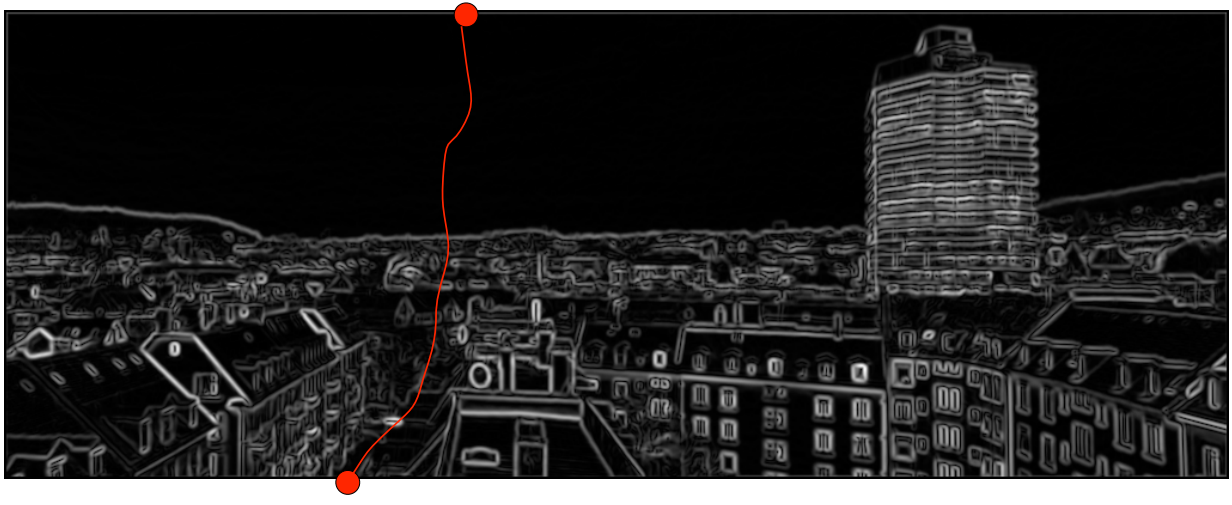
energy of sample image

thanks to Jason Lawrence for gradient software

BEST SEAM HAS LOWEST ENERGY



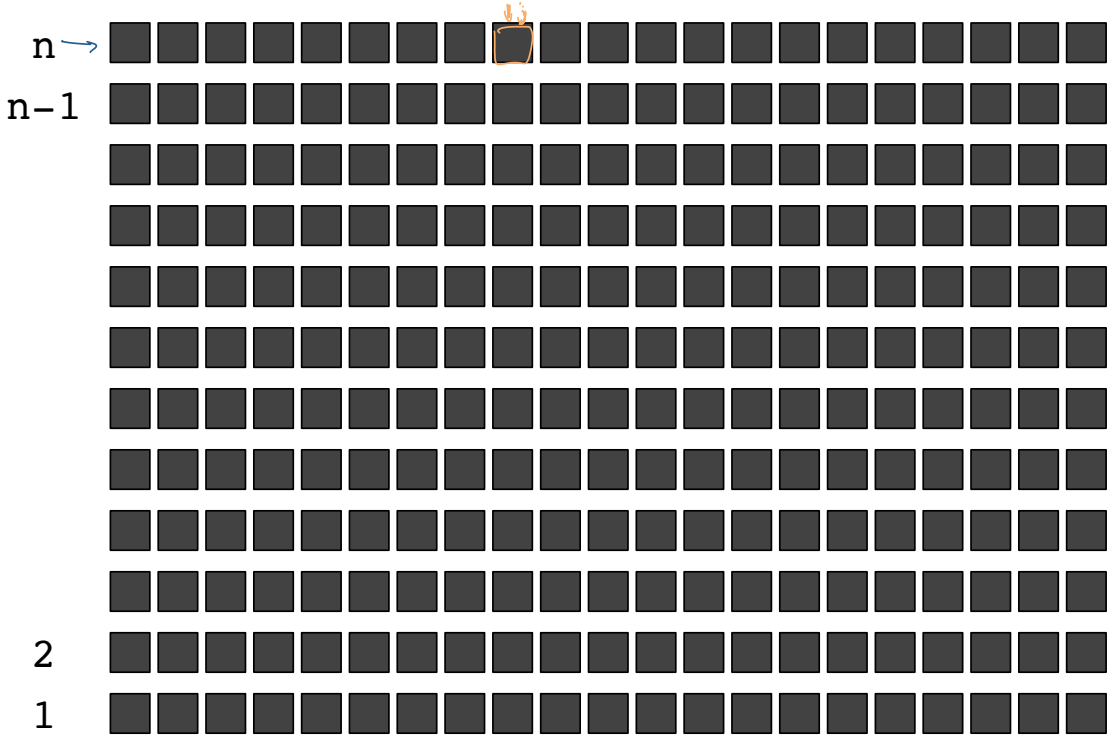
FINDING LOWEST ENERGY SEAM?



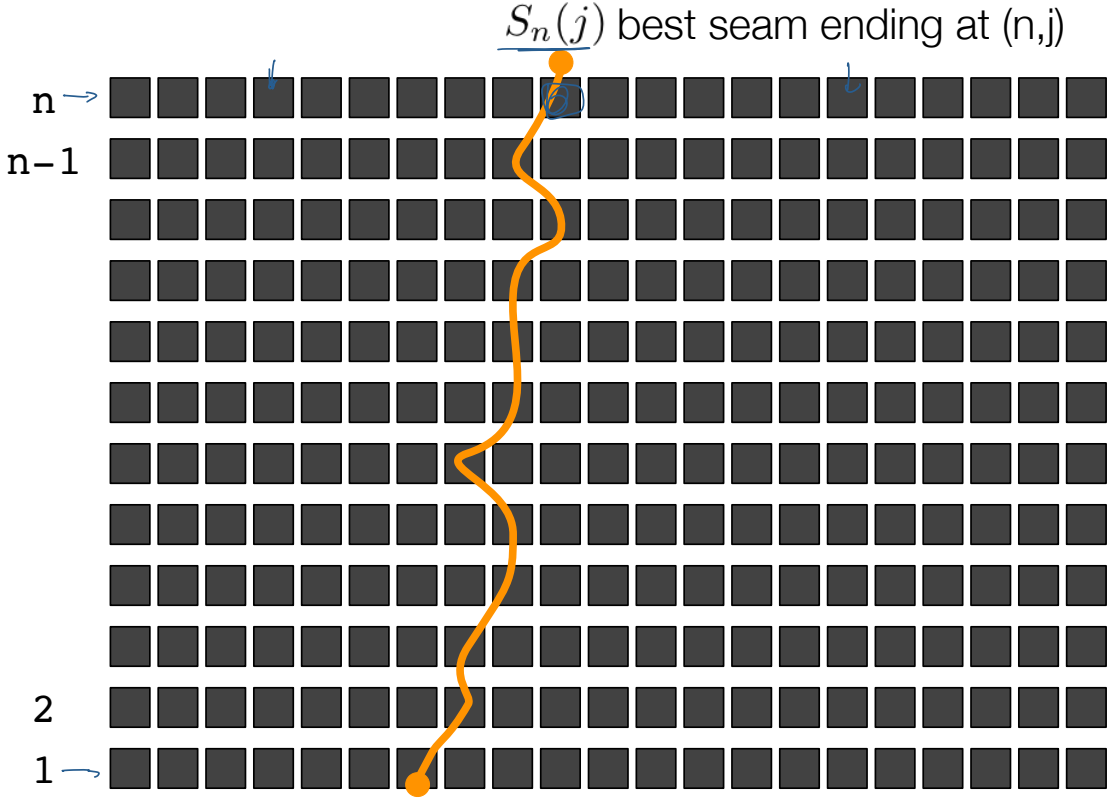
Define a variable:

$$S_i(j)$$

definition: $S_n(j)$



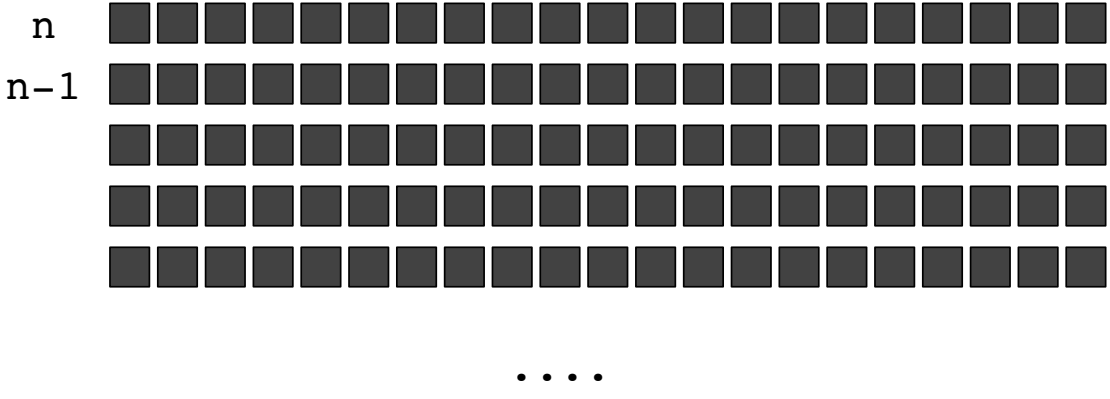
definition:



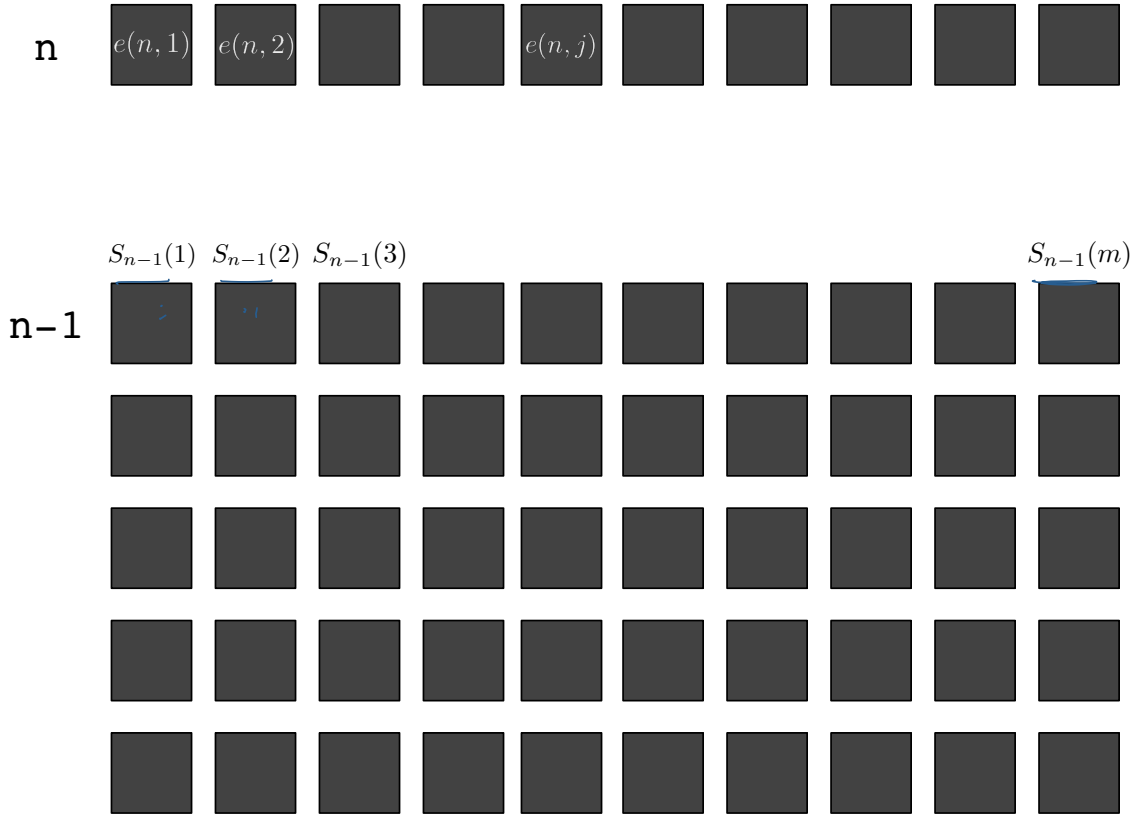
BEST SEAM TO DELETE HAS TO BE THE BEST AMONG

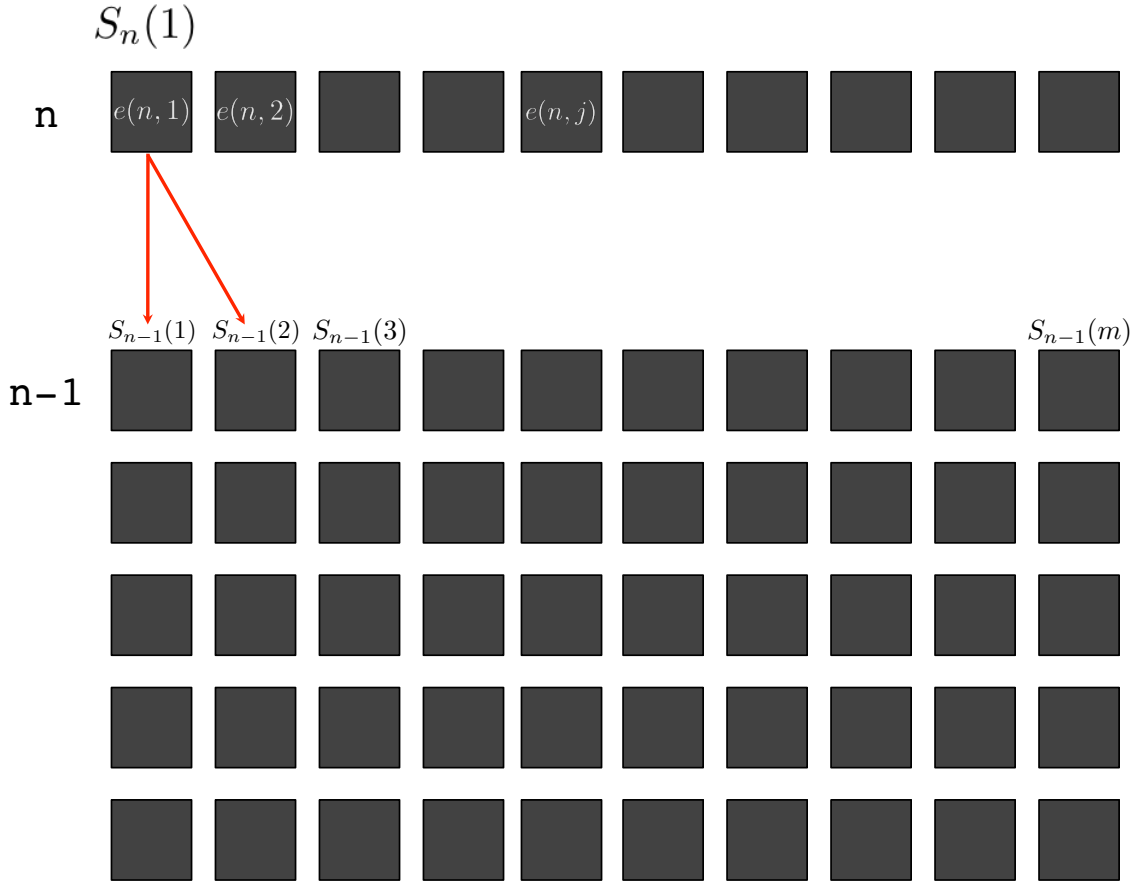
$$S_n(1), S_n(2), \ldots, S_n(m)$$

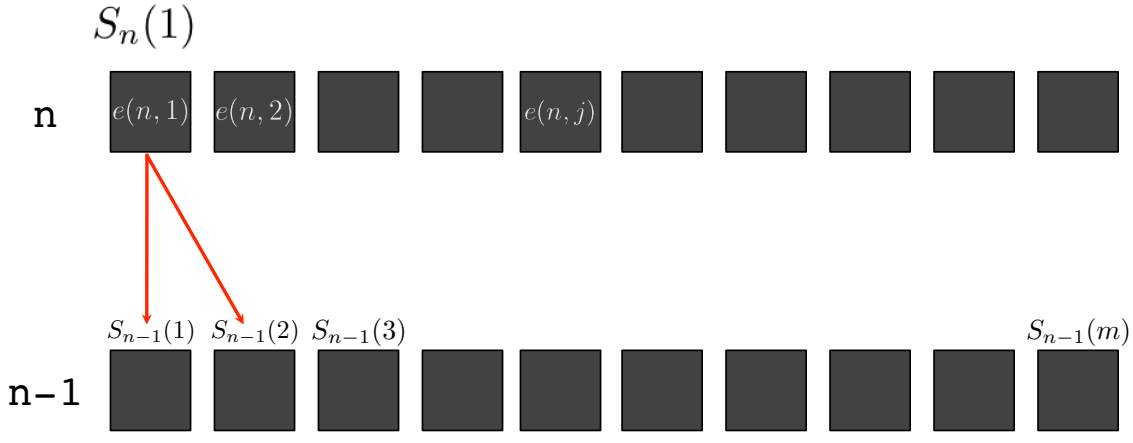
IDEA: COMPUTE + COMPARE



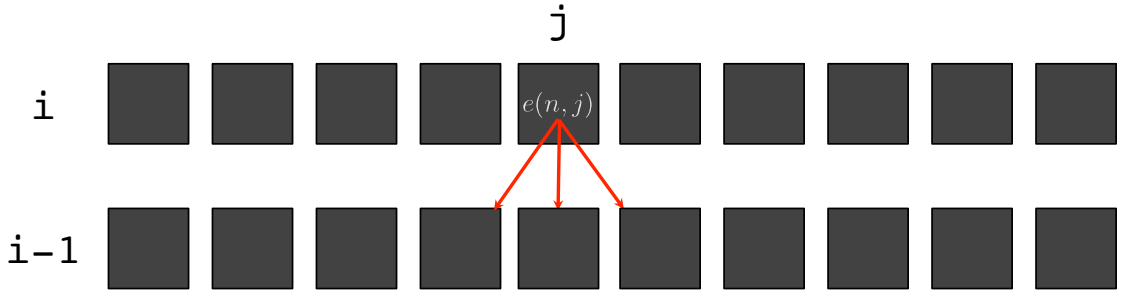
IMAGINE YOU HAVE THE SOLUTION TO THE FIRST N-1 ROWS


n
$e(n,1)$
$e(n,2)$
$e(n,j)$
n−1
$S_{n-1}(1)$
$S_{n-1}(2)$
$S_{n-1}(3)$
$S_{n-1}(m)$

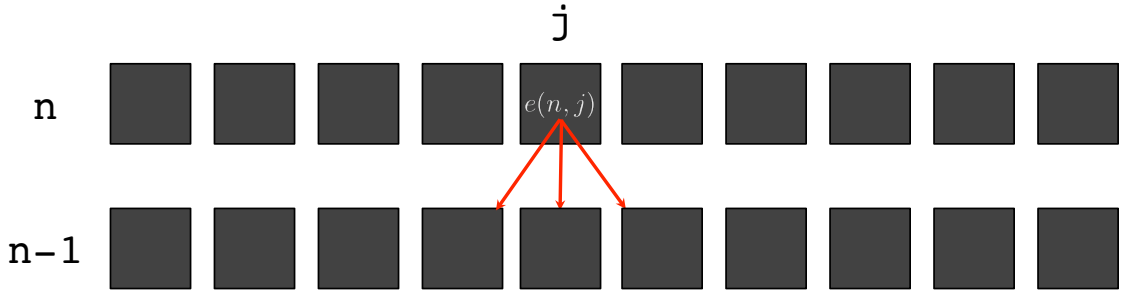

$S_n(1)$
n
$e(n,1)$
$e(n,2)$
$e(n,j)$
$S_{n-1}(1)$
$S_{n-1}(2)$
$S_{n-1}(3)$
$S_{n-1}(m)$
n−1



$$S_n(1) = e(n,1) + \min\{S_{n-1}(1), S_{n-1}(2)\}$$



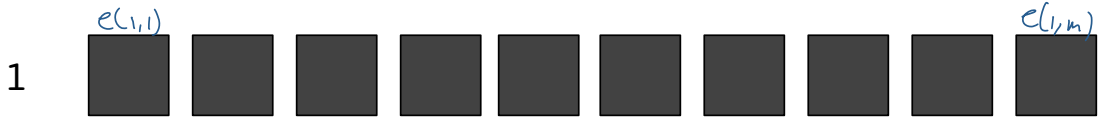
$$S_i(j) =$$



$$S_i(j) = e(i,j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

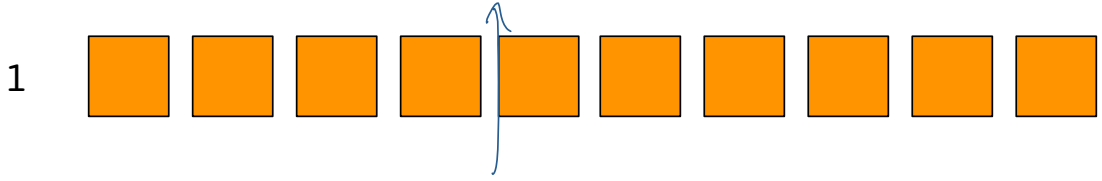
ALGORITHM

start at bottom of picture



ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1, i)$

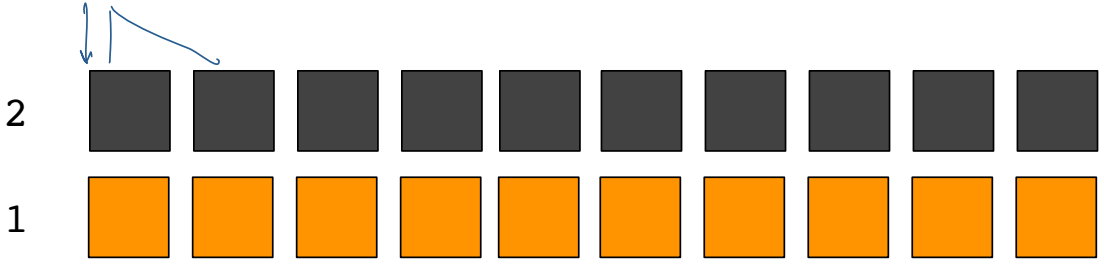


ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1, i)$

for `i=2 to n` use formula to compute $S_{i+1}(\cdot)$

$$S_i(j) = e(i, j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

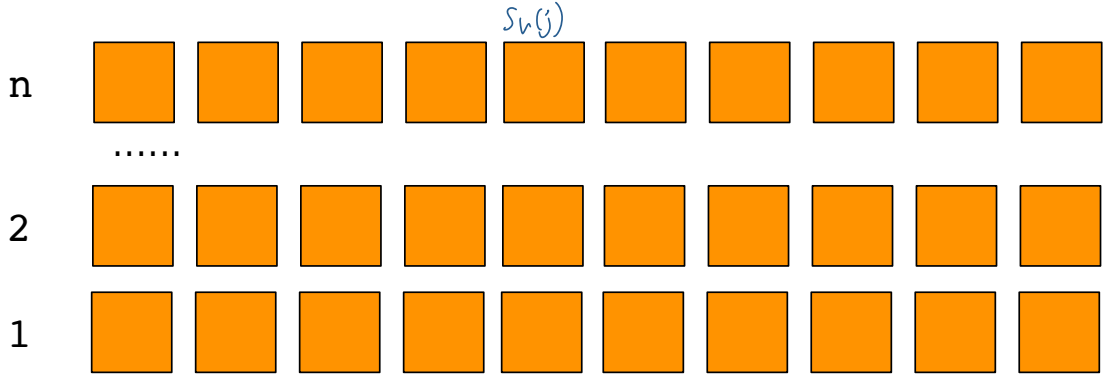


ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1,i)$

for `i=2,n` use formula to compute $S_{i+1}(\cdot)$

$$S_i(j) = e(i,j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$



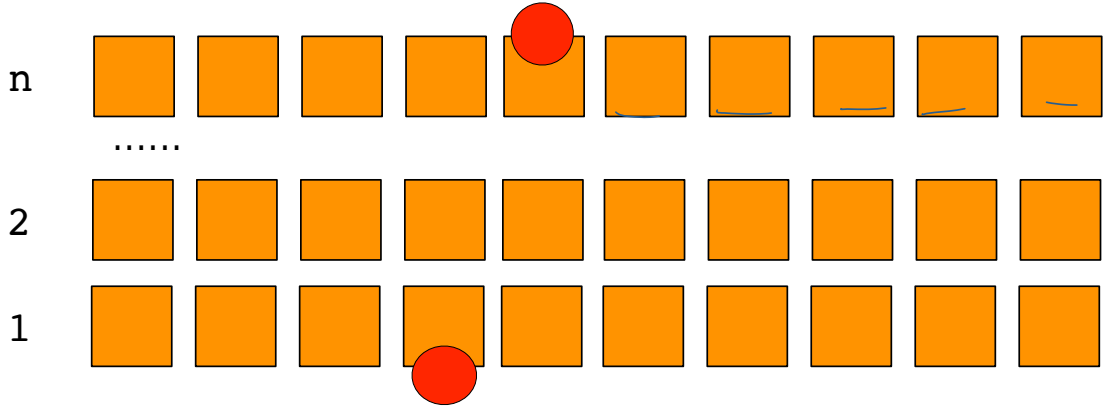
ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1, i)$

for `i=2,n` use formula to compute $S_{i+1}(\cdot)$

$$S_i(j) = e(i, j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

pick best among top row, backtrack.



RUNNING TIME

start at bottom of picture. initialize $S_1(i) = e(1, i)$

for `i=2,n` use formula to compute $S_{i+1}(\cdot)$

$$S_i(j) = e(i, j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

pick best among top row, backtrack.