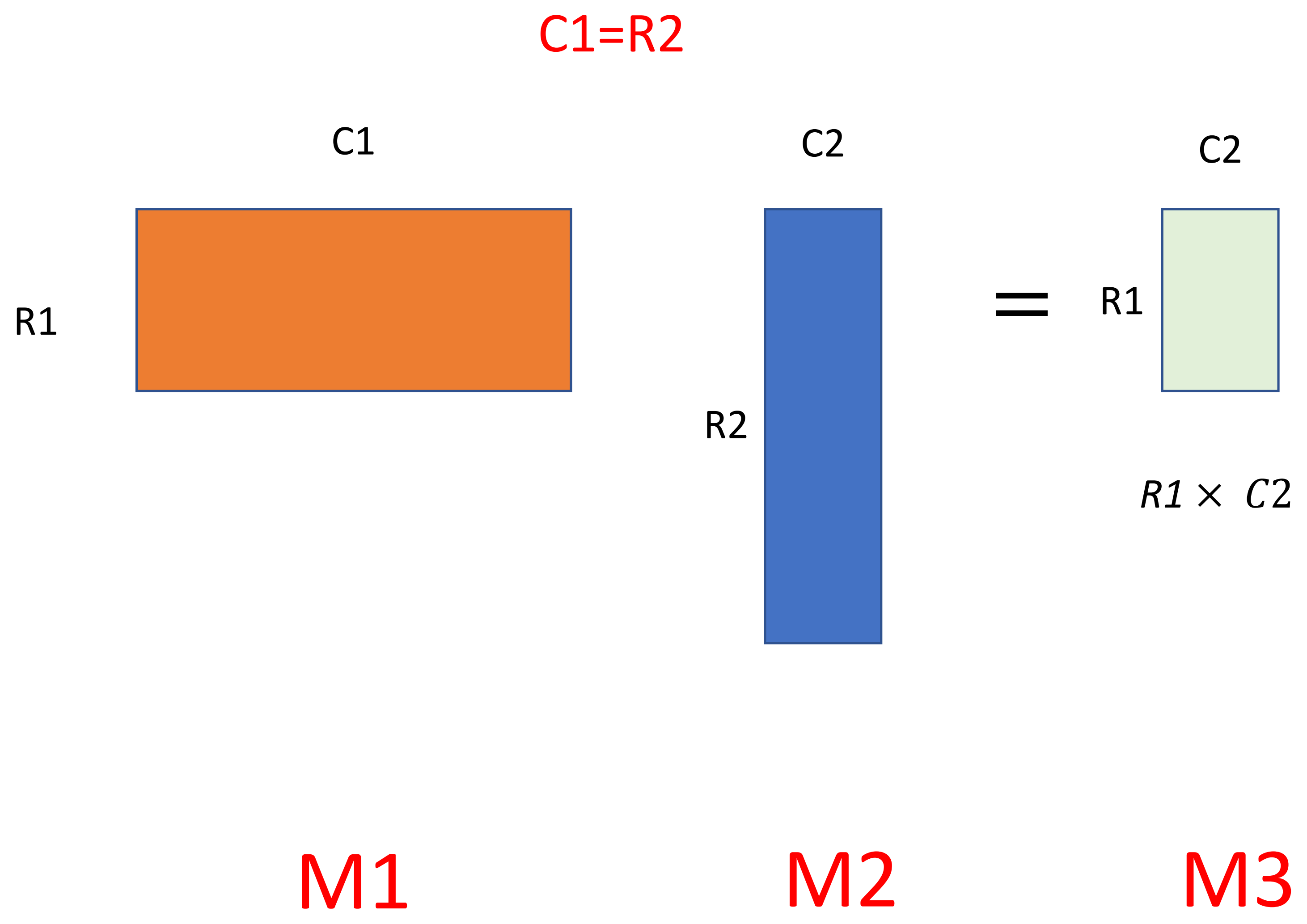
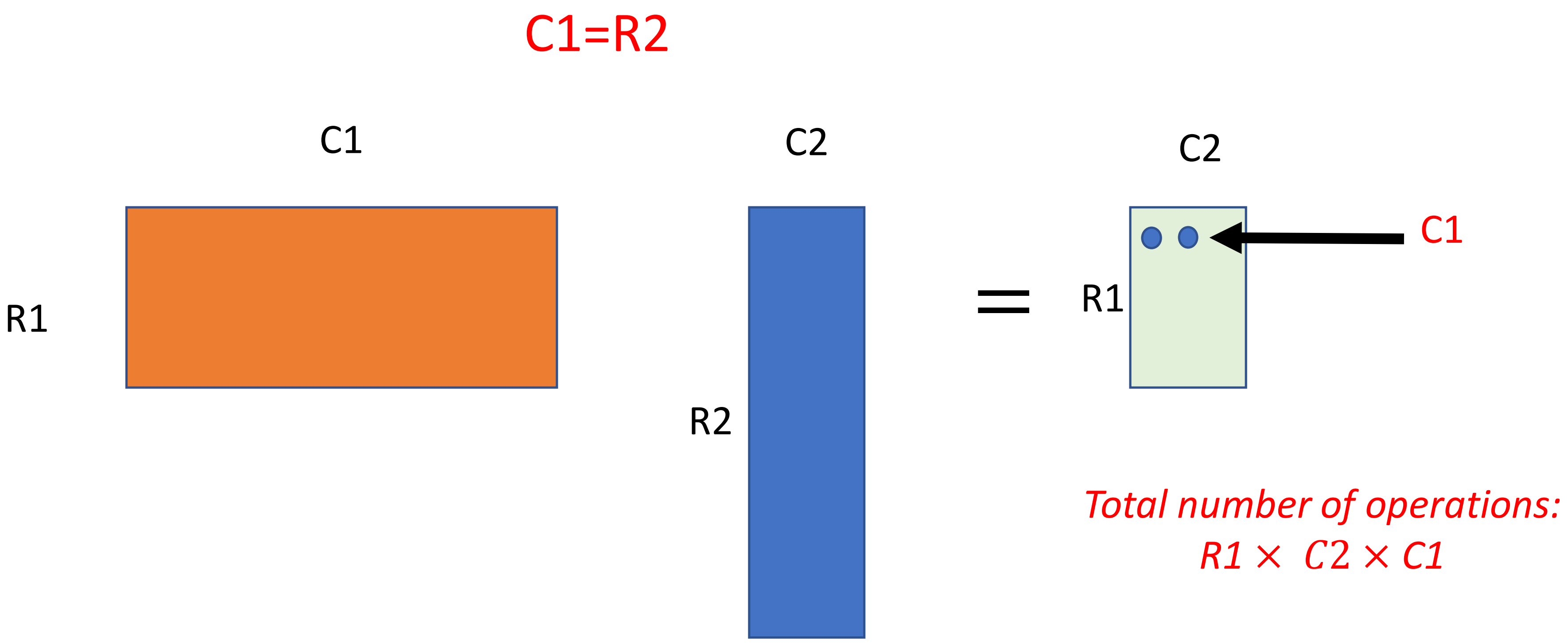
Dynamic Programming

lecture 2


C1=R2
C1
C2
C2
R1
=
R1
R2
R1 × C2
M1
M2
M3



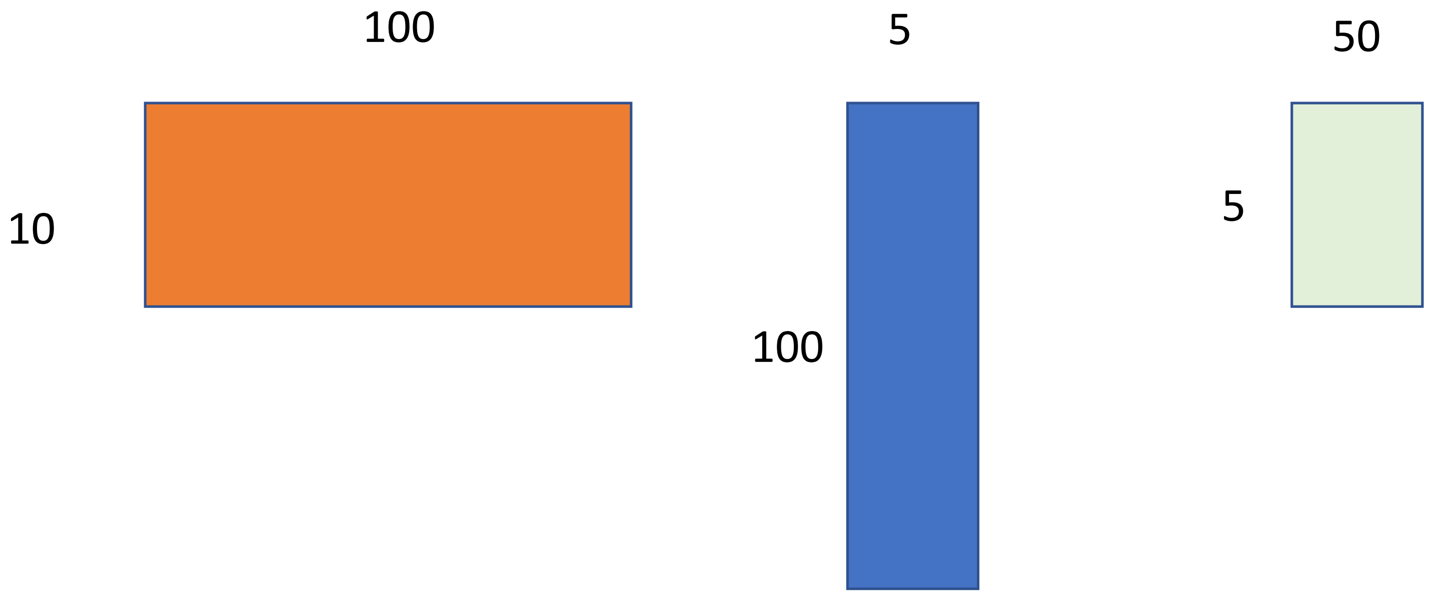
C1=R2
C1
R1
C2
R2
=
C2
R1
C1
Total number of operations:
$R1 \times C2 \times C1$

$$A_1 \cdot A_2 \cdot A_3$$

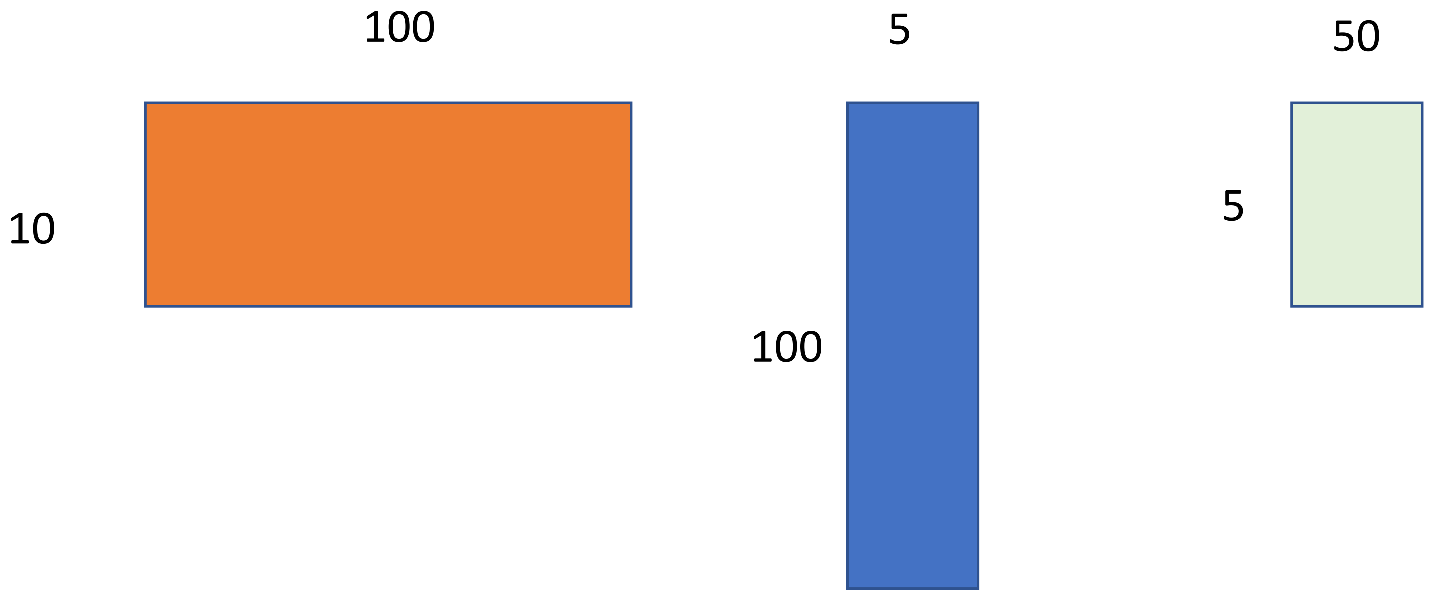
Associative

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

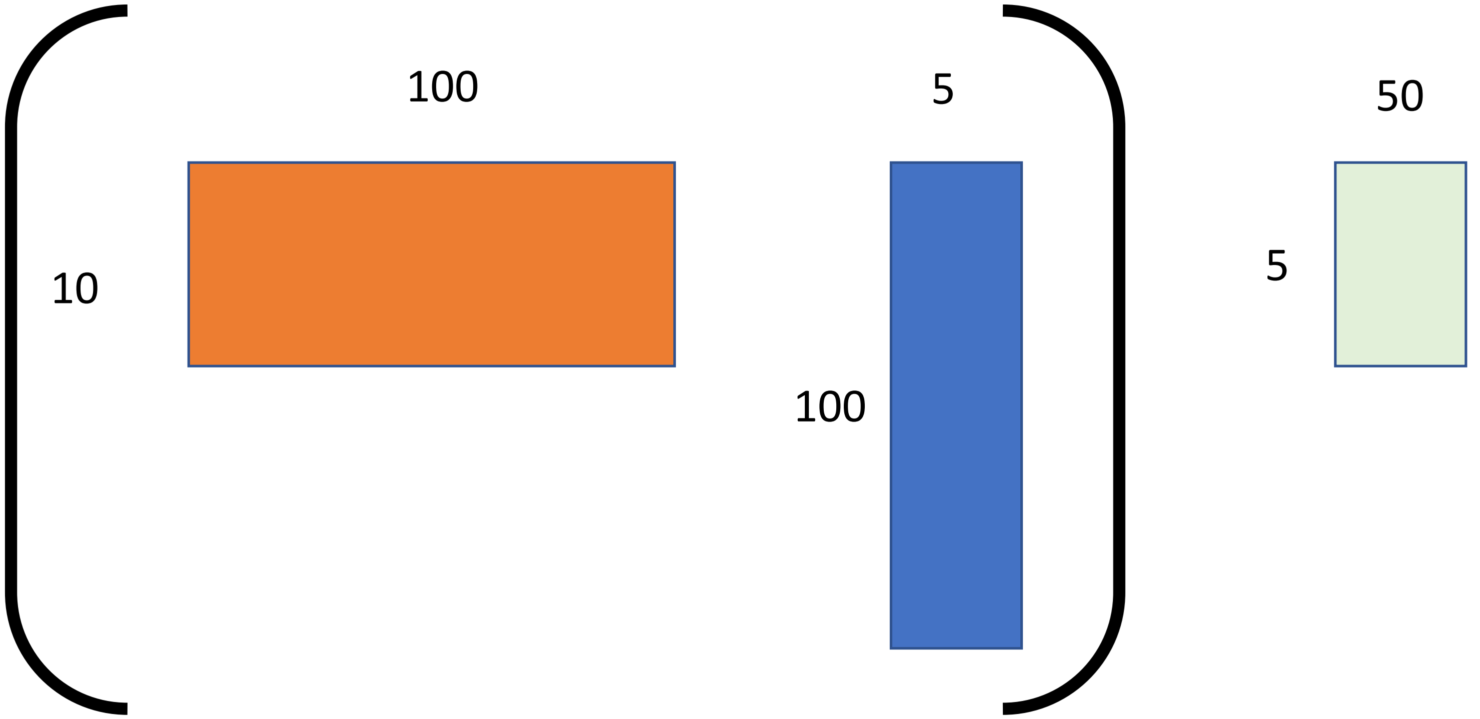
$$A_1 \cdot A_2 \cdot A_3$$



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



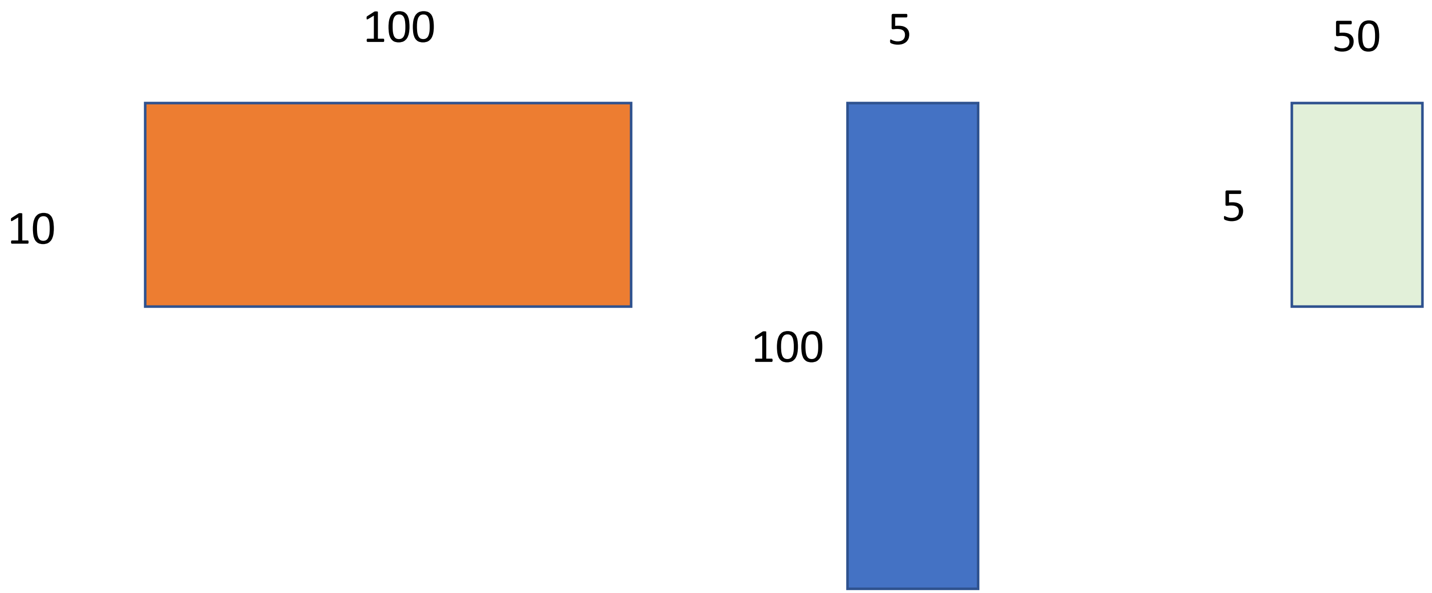
$$(A_1 \cdot A_2) \cdot A_3$$



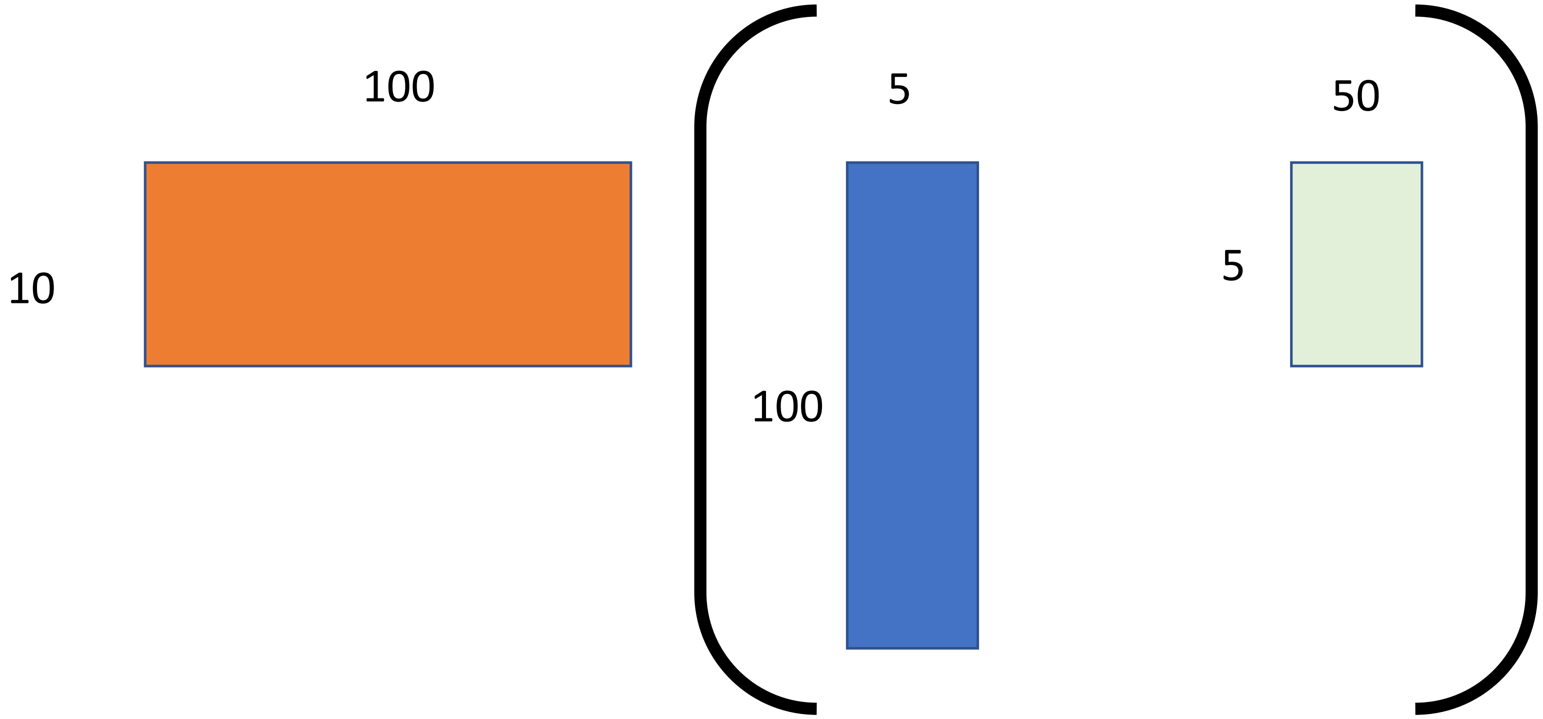
7500

$$10.100.5 + 10.5.50$$

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



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



75000

$$100.5.50 + 10.100.50$$

Order Matters

How many ways to multiply:

$$A_1 \cdot A_2 \cdot A_3 \cdot \cdot \cdot \cdot \; A_n$$

N-1 multiplication

P(n): number of ways to multiply the n matrices

How many ways to multiply:

$$A_1 \cdot A_2 \cdot A_3 \cdot \cdot \cdot \cdot \; A_n$$

P(n): number of ways to multiply the n matrices

$P(n) = P(1).P(n-1) +$

$$\boxed{A_1} \cdot \boxed{A_2 \cdot A_3 \cdot \cdot \cdot \cdot \; A_n}$$

How many ways to multiply:

$$A_1 \cdot A_2 \cdot A_3 \cdot \cdot \cdot \cdot \; A_n$$

P(n): number of ways to multiply the n matrices

P(n) = P(1).P(n-1) + P(2).P(n-2) +

$$\boxed{A_1 \cdot A_2} \cdot \boxed{A_3 \cdot \cdot \cdot \cdot \; A_n}$$

How many ways to multiply:

$$A_1 \cdot A_2 \cdot A_3 \cdot \cdot \cdot \cdot \; A_n$$

P(n): number of ways to multiply the n matrices

P(n) = P(1).P(n-1) + P(2).P(n-2) + P(3).P(n-3) +

$$\boxed{A_1 \cdot A_2 \cdot A_3} \cdot \boxed{A_4 \cdot \cdot \cdot \; A_n}$$

How many ways to multiply:

$$A_1 \cdot A_2 \cdot A_3 \cdot \cdot \cdot \cdot \; A_n$$

P(n): number of ways to multiply the n matrices

P(n) = P(1).P(n-1) + P(2).P(n-2) + P(3).P(n-3) + ……… + P(n-1)P(1)

$$\boxed{A_1 \cdot A_2 \cdot A_3 \cdots\cdots A_{n-1}} \cdot \boxed{A_n}$$

How many ways to multiply:

$$A_1 \cdot A_2 \cdot A_3 \cdot \cdot \cdot \cdot \; A_n$$

P(n): number of ways to multiply the n matrices

P(n) = P(1).P(n-1) + P(2).P(n-2) + P(3).P(n-3) + ……… + P(n-1)P(1)

$$= \sum_{i=1}^{n-1} P(i).P(n-i) \approx 4^n$$

Optimal Way to Compute

$$A_1 \cdot A_2 \cdot A_3 \ldots\ldots A_l \cdot A_{l+1} \cdot\cdot\cdot A_n$$

Optimal Way to Compute

$$\overset{C1}{{}_{R1}A_1} \cdot \overset{C2}{{}_{R2}A_2} \cdot \overset{C3}{{}_{R3}A_3} \cdots\cdots \overset{Cl}{{}_{Rl}A_l} \cdot \overset{C_{l+1}}{{}_{R_{l+1}}A_{l+1}} \cdot \;\cdot\; \cdot \overset{Cn}{{}_{Rn}A_n}$$

B[1,n]= smallest number of operations needed to multiply the chain

Optimal Way to Compute

$$\overset{C1}{{}^{R1}A_1} \cdot \overset{C2}{{}^{R2}A_2} \cdot \overset{C3}{{}^{R3}A_3} \cdots\cdots \overset{Cl}{{}^{Rl}A_l} \cdot \overset{C_{l+1}}{{}^{R_{l+1}}A_{l+1}} \cdot\ \cdot\ \cdot \overset{Cn}{{}^{Rn}A_n}$$

B[1,n]= smallest number of operations needed to multiply the chain

Optimal Way to Compute

C1 C2 C3 Cl C_{l+1} Cn

R1 A_1 $\cdot$ R2 A_2 $\cdot$ R3 $A_3 \ldots\ldots$ Rl A_l $\cdot$ R_{l+1} $A_{l+1} \cdot \cdot \cdot$ Rn A_n

Optimal last step: A[1…l] . A[l+1,….n]

B[1,n]= smallest number of operations needed to multiply the chain

$B[1,n] = B[1,l] + B[l+1,n] + R_1.C_l.C_{l+1}$

Optimal Way to Compute

$$\underbrace{{}^{R1}\overset{C1}{A_1} \cdot {}^{R2}\overset{C2}{A_2} \cdot {}^{R3}\overset{C3}{A_3} \cdots\cdots {}^{Rl}\overset{Cl}{A_l}} \cdot \underbrace{{}^{R_{l+1}}\overset{C_{l+1}}{A_{l+1}} \cdot\ \cdot\ \cdot\ {}^{Rn}\overset{Cn}{A_n}}$$

Optimal last step: A[1…l] . A[l+1,….n]

B[1,n]= smallest number of operations needed to multiply the chain

$$\mathbf{B[1,n]= \ B[1,l] + B[l+1,n] + R_1.C_l.C_{l+1}}$$

How many choices we have for l?

$l \in [1,n-1]$

Optimal Way to Compute

$$ {}^{R_1}\overset{C_1}{A_1} \cdot {}^{R_2}\overset{C_2}{A_2} \cdot {}^{R_3}\overset{C_3}{A_3} \cdots\cdots {}^{R_l}\overset{C_l}{A_l} \cdot {}^{R_{l+1}}\overset{C_{l+1}}{A_{l+1}} \cdot \;\cdot\; \cdot {}^{R_n}\overset{C_n}{A_n} $$

B[1,n]= smallest number of operations needed to multiply the chain

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]**
$R_1C_1C_n$	$R_1C_2C_n$			$R_1C_{n-2}C_n$	$R_1C_{n-1}C_n$

Which Order to Solve?

$$A_1 \cdot A_2 \cdot A_3 \cdot \, . \, . \, . \, A_{n-1} \cdot A_n$$

B(i,i)=0

$$B(i,j) = \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\, j) + R_i C_k C_j$$

$$R_1\,\overset{C_1}{A_1} \cdot R_2\,\overset{C_2}{A_2} \cdot R_3\,\overset{C_3}{A_3} \cdots\cdots R_k\,\overset{C_k}{A_k}\; R_{k+1} \cdot \overset{C_{k+1}}{A_{k+1}} \cdots R_{n-1}\,\overset{C_{n-1}}{A_{n-1}} \cdot R_n\,\overset{C_n}{A_n}$$

Which Order to Solve?

$A_1 \cdot A_2 \cdot A_3 \cdot \,.\,.\,.\, A_{n-1} \cdot A_n$

$B(i,i)=0$

$i=2$

$j=n-1$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_iC_kC_j$$

$${}_{R_1}\overset{C_1}{A_1} \cdot \boxed{{}_{R_2}\overset{C_2}{A_2} \cdot {}_{R_3}\overset{C_3}{A_3} \cdots\cdots {}_{R_k}\overset{C_k}{A_k}} \cdot \boxed{{}_{R_{k+1}}\overset{C_{k+1}}{A_{k+1}} \cdots {}_{R_{n-1}}\overset{C_{n-1}}{A_{n-1}}} \cdot {}_{R_n}\overset{C_n}{A_n}$$

Which Order to Solve?

$$A_1 \cdot A_2 \cdot A_3 \cdot \,.\,.\,.\, A_{n-1} \cdot A_n$$

$B(i,i)=0$

$$B(i,j) = \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$

$i=2$

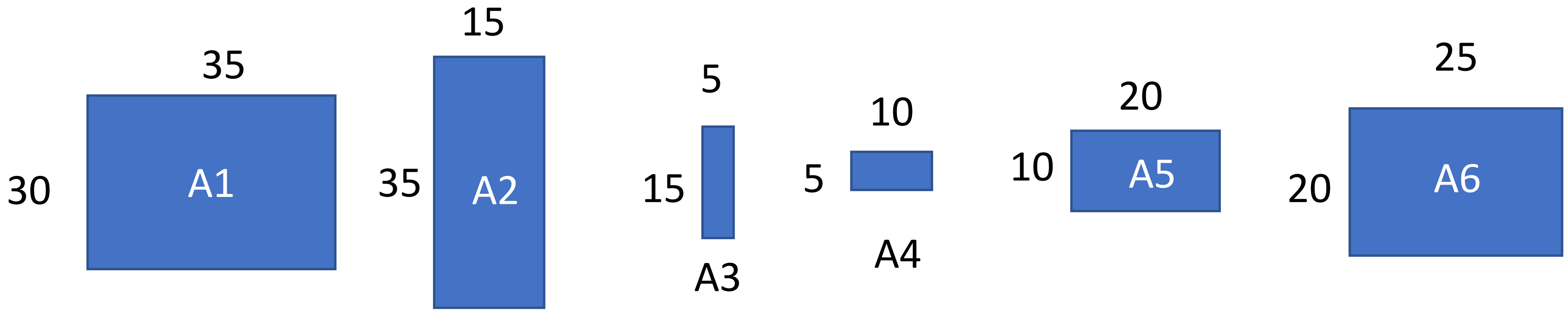
$j=n-1$

$K=3$

C_1 C_2 C_3 C_4 C_{n-1} C_n

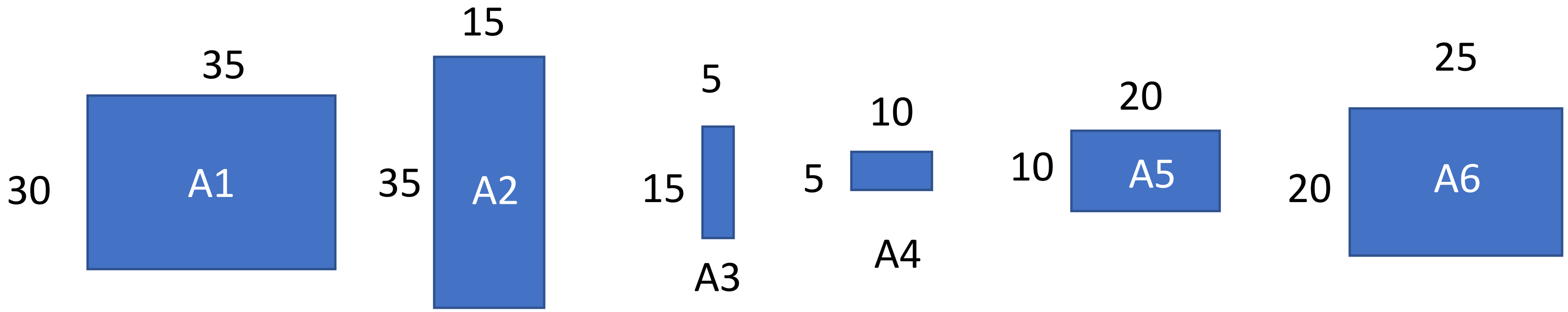
R_1 R_2 R_3 R_4 R_{n-1} R_n

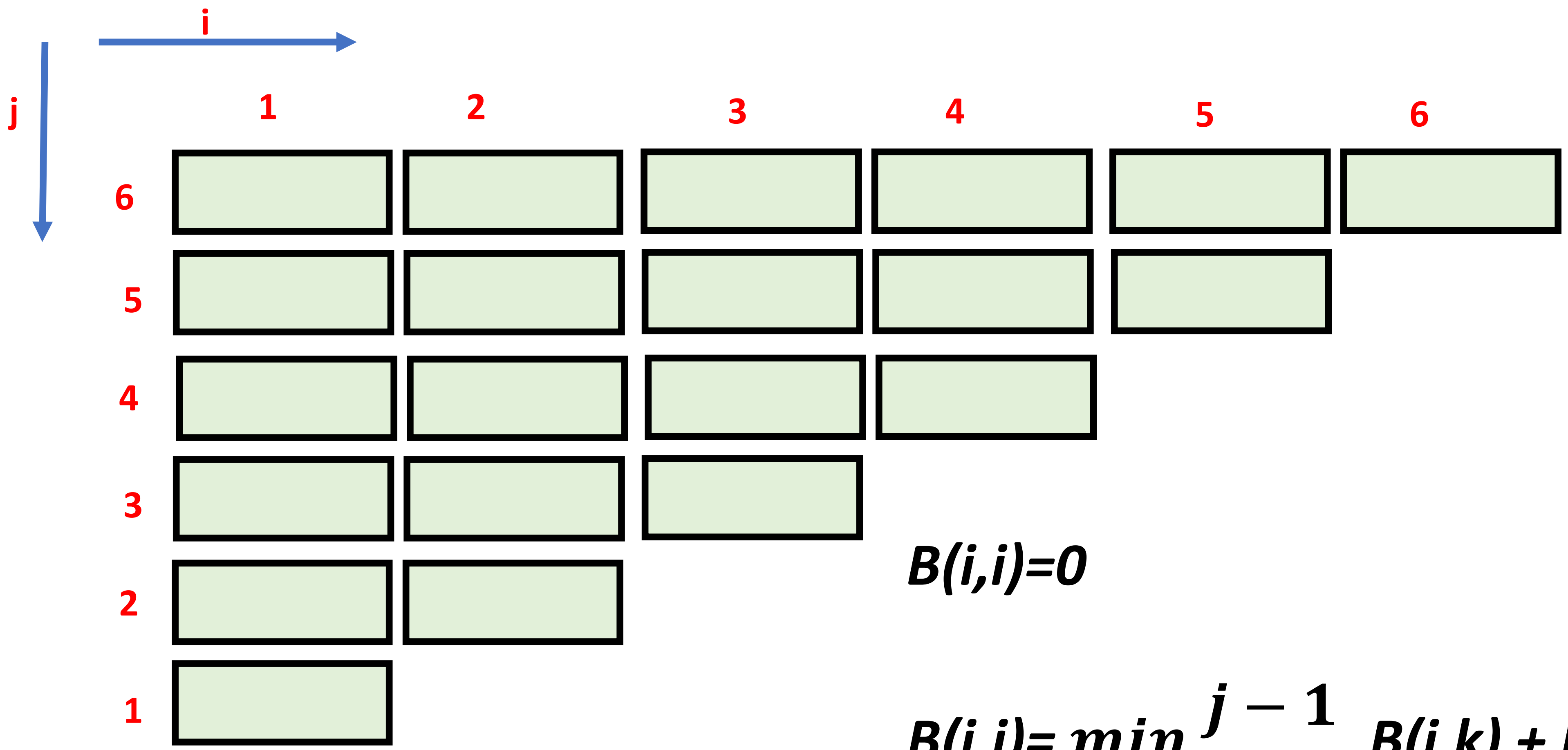
$$A_1 \cdot \boxed{A_2 \cdot A_3} \cdot \boxed{A_4 \cdots\cdots A_{n-1}} \cdot A_n$$



$$B(i,i)=0$$

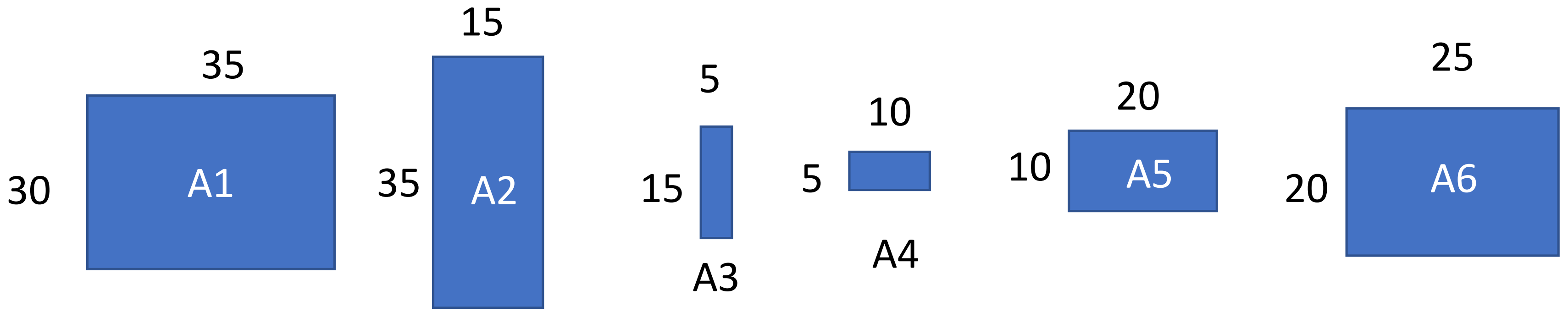
$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\, ,\, j) + R_i C_k C_j$$





$$B(i,i)=0$$

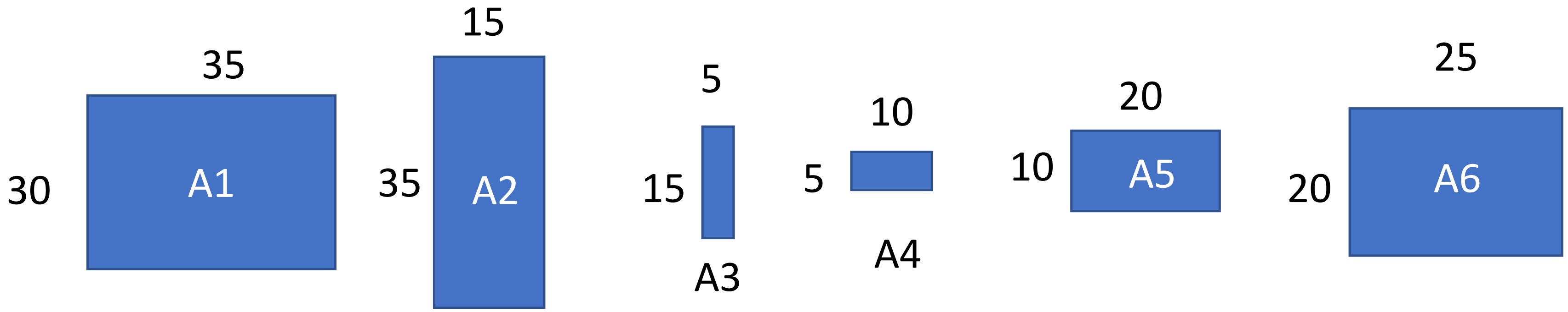
$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$



j \ i	1	2	3	4	5	6
6						0
5					0	
4				0		
3			0			
2		0				
1	0					

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$



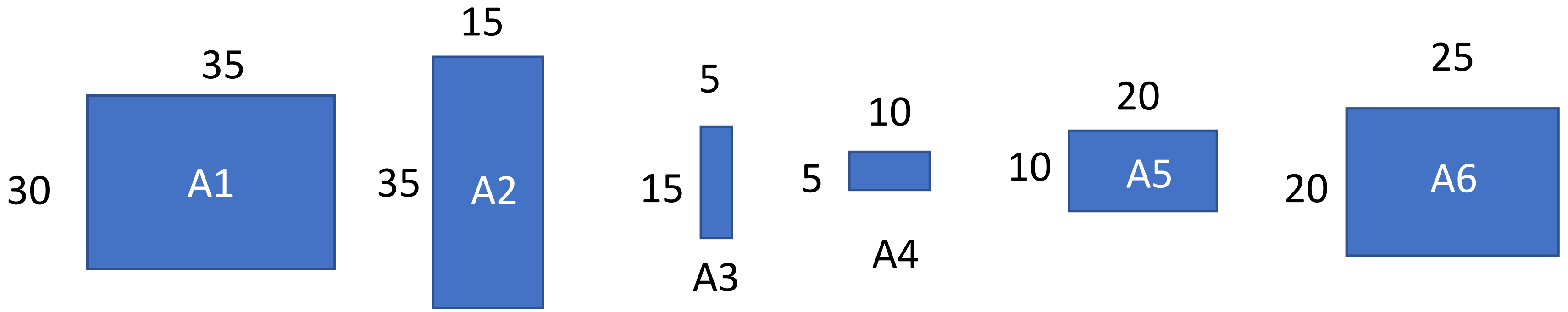
i

j

	1	2	3	4	5	6
6	*B(1,6)*					0
5					0	
4				0		
3			0			
2		0				
1	0					

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_iC_kC_j$$

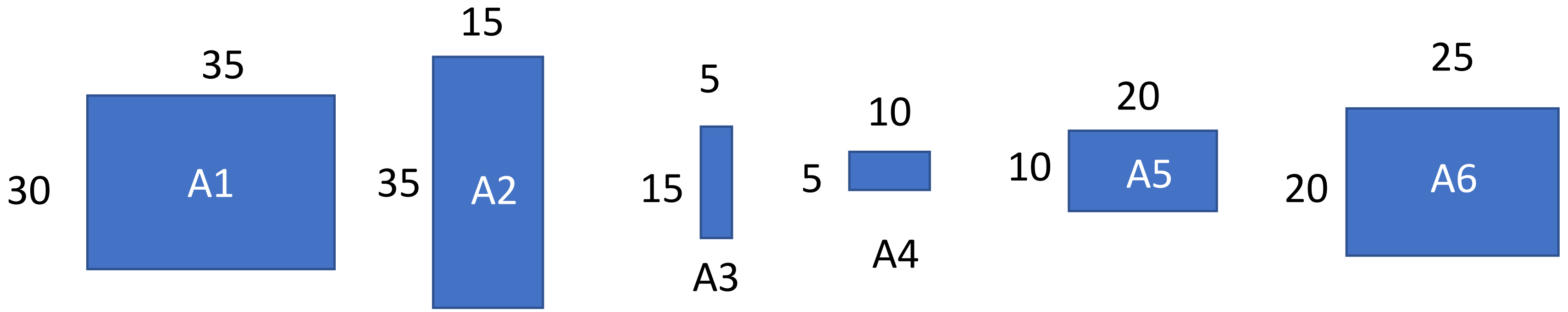


j \ i	1	2	3	4	5	6
6						0
5					0	
4				0		
3			0			
2		0				
1	0					

$$B(1,2)=B(1,1)+B(2,2)+30.35.15$$

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$

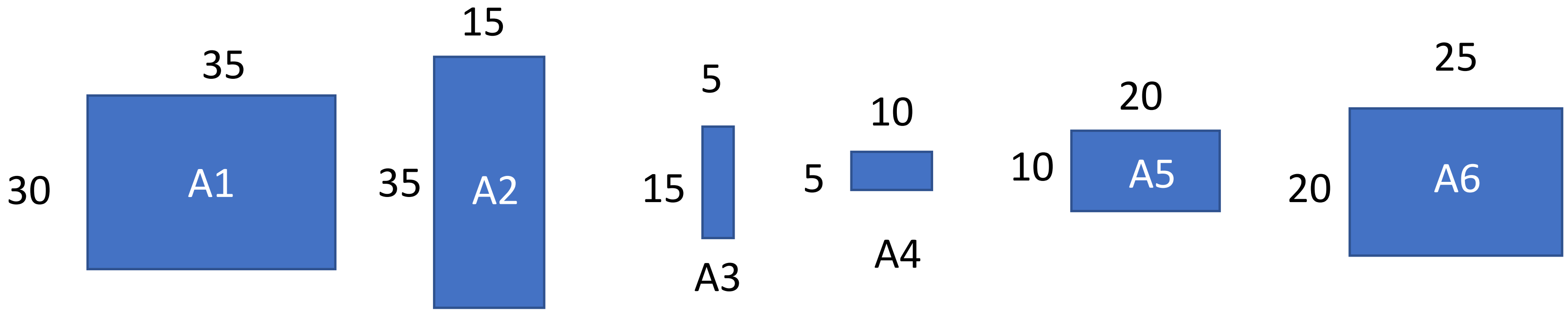


j \ i	1	2	3	4	5	6
6						0
5					0	
4				0		
3			0			
2	15750	0				
1	0					

$$B(1,2)=B(1,1)+B(2,2)+30.35.15$$

$$B(i,i)=0$$

$$B(i,j)=\min_{k=i}^{j-1} B(i,k)+B(k+1,j)+R_i C_k C_j$$

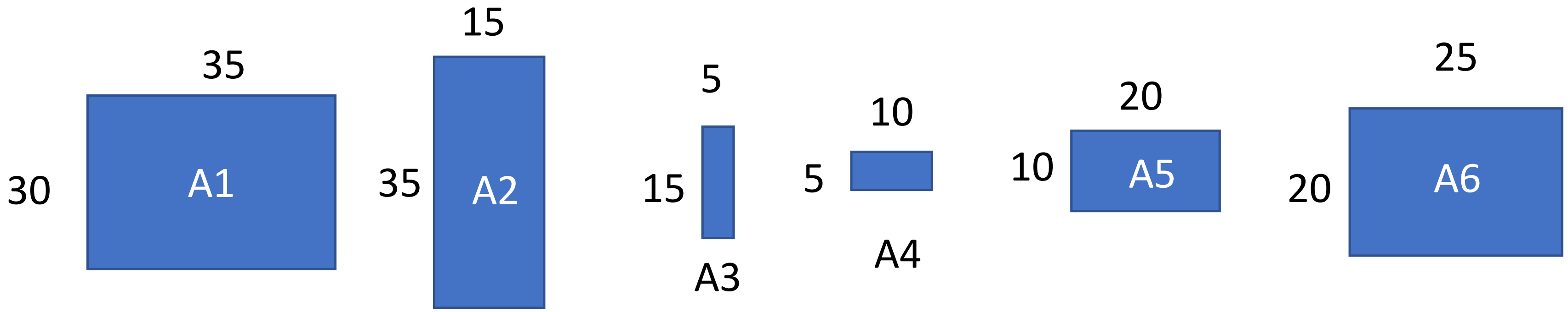


j \ i	1	2	3	4	5	6
6						0
5					0	
4				0		
3			0			
2	15750	0				
1	0					

$$B(2,3)=B(2,2)+B(3,3)+35.15.5$$

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_iC_kC_j$$



i

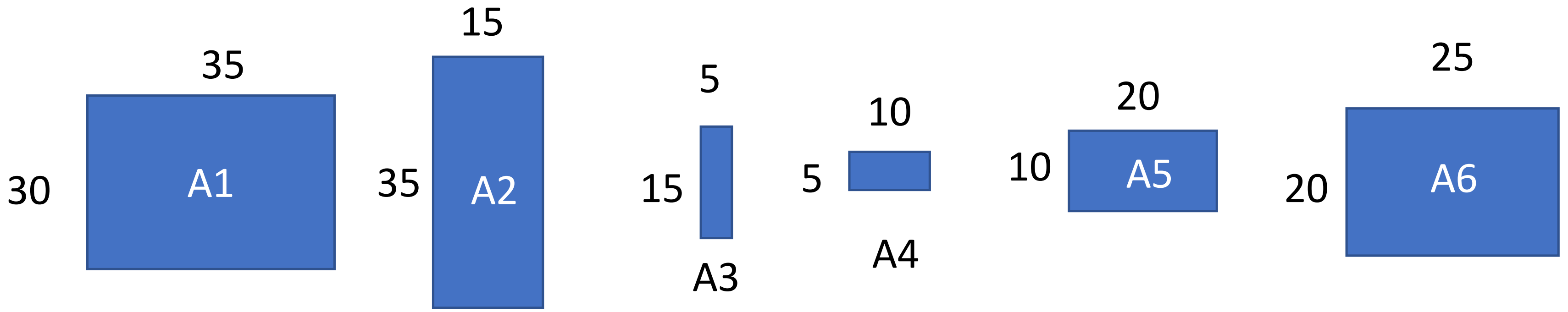
j

j \ i	1	2	3	4	5	6
6						0
5					0	
4				0		
3		2625	0			
2	15750	0				
1	0					

$$B(2,3)=B(2,2)+B(3,3)+35.15.5$$

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\, j) + R_i C_k C_j$$

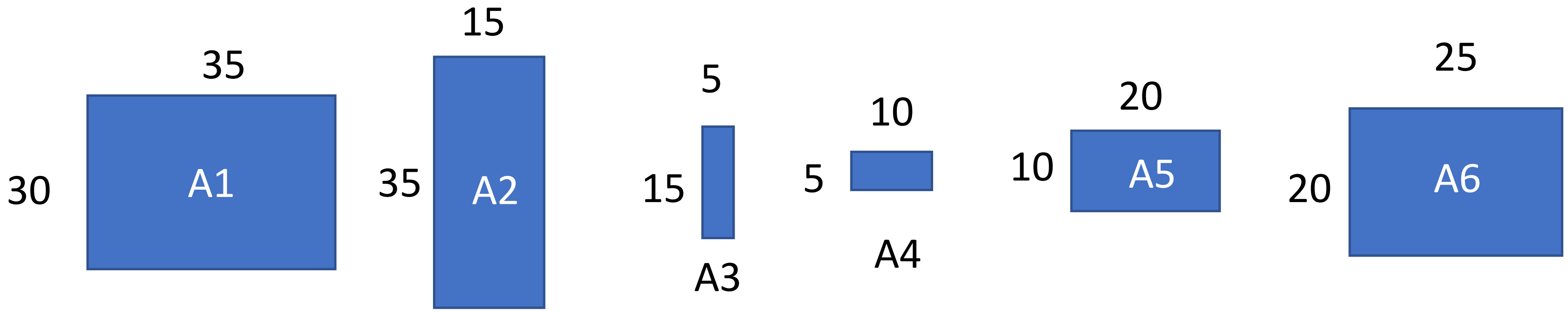


j \ i	1	2	3	4	5	6
6						0
5					0	
4				0		
3		2625	0			
2	15750	0				
1	0					

$$B(3,4)=B(3,3)+B(4,4)+15.5.10$$

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$



i

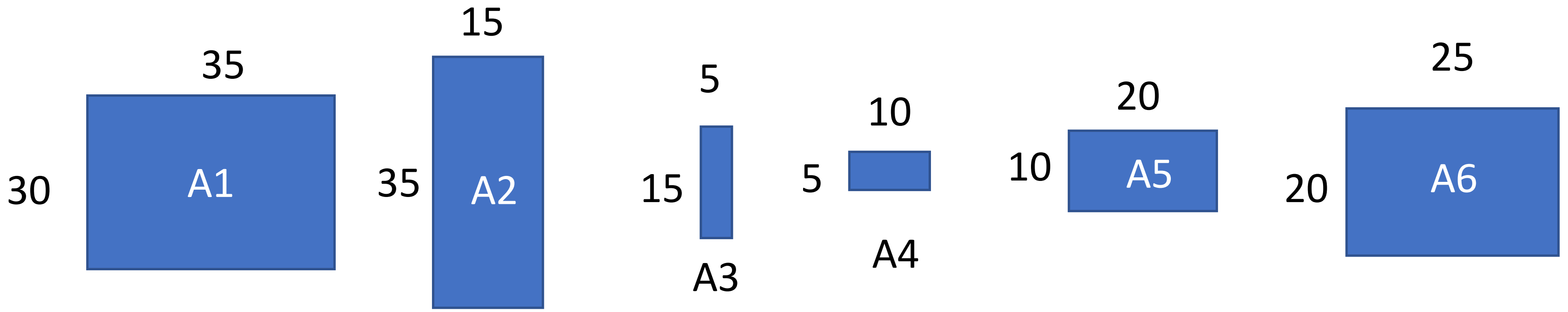
j

	1	2	3	4	5	6
6						0
5					0	
4			750	0		
3		2625	0			
2	15750	0				
1	0					

$$B(3,4)=B(3,3)+B(4,4)+15.5.10$$

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_iC_kC_j$$



i

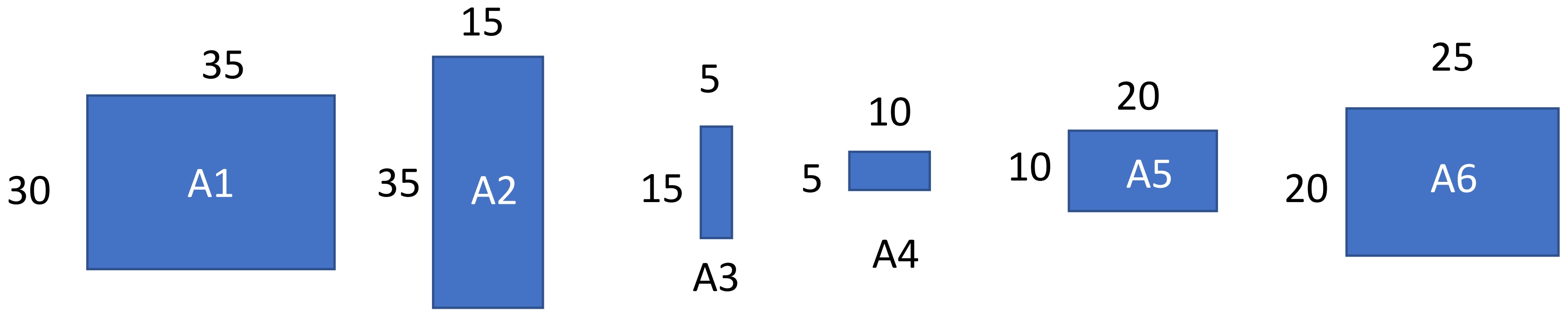
j	1	2	3	4	5	6
6					5000	0
5				1000	0	
4			750	0		
3		2625	0			
2	15750	0				
1	0					

$$B(4,5)=B(4,4)+B(5,5)+5.10.20$$

$$B(5,6)=B(5,5)+B(6,6)+10.20.25$$

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$

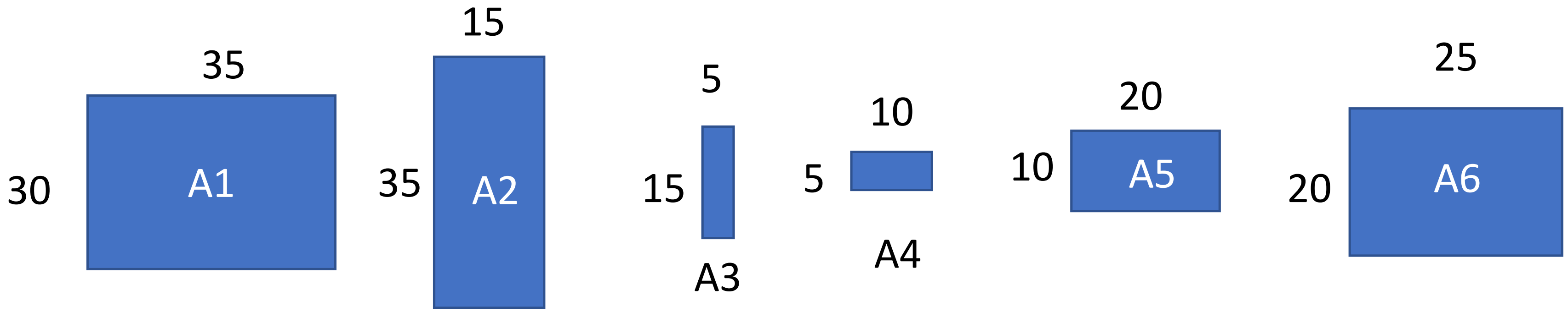


j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4			750	0		
3		2625	0			
2	15750	0				
1	0					

B(1,3)

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$

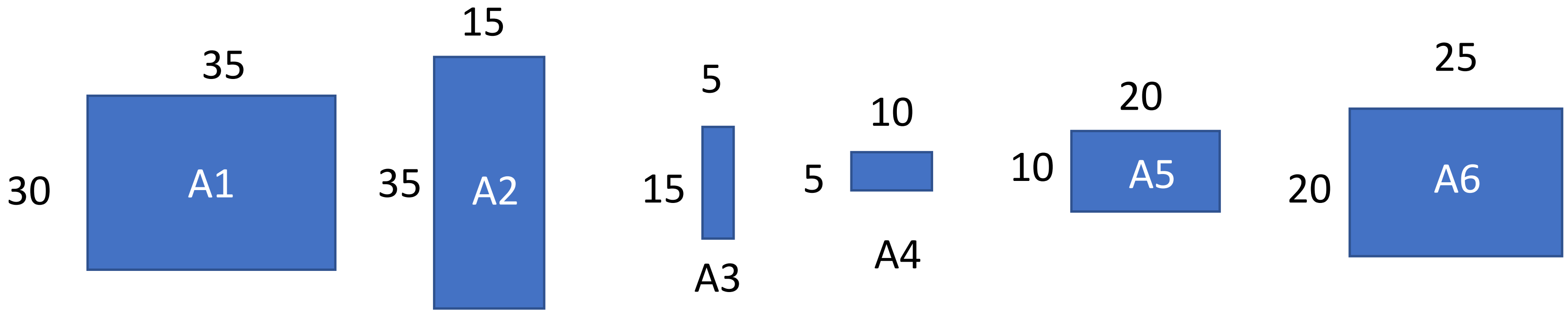


j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4			750	0		
3	$\boxed{}$	2625	0			
2	15750	0				
1	0					

$$B(1,3)=\min \begin{cases} K=2 \\ B(1,2)+B(3,3)+30.15.5 \\ B(1,1)+B(2,3)+30.35.5 \\ K=1 \end{cases}$$

$$B(i,i)=0$$

$$B(i,j)=\min_{k=i}^{j-1} B(i,k)+B(k+1\,,\,j)+R_i C_k C_j$$

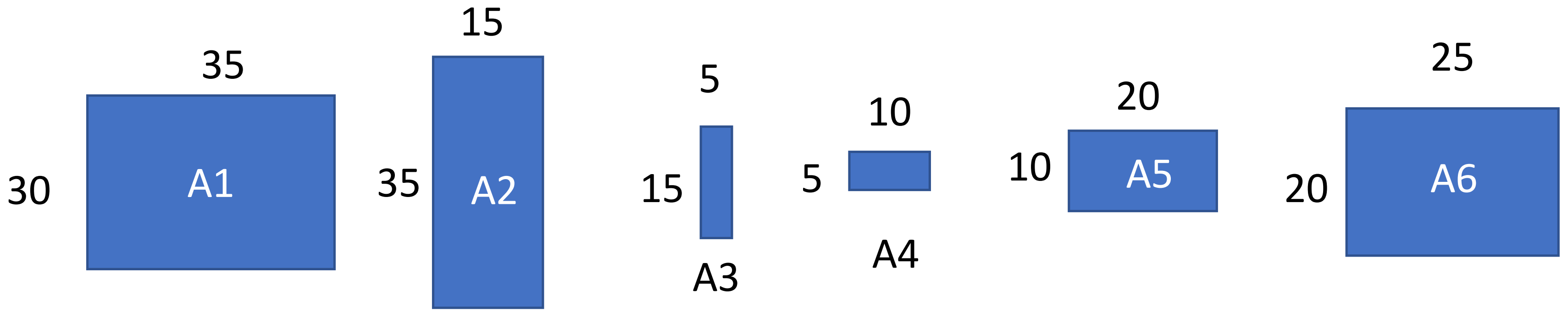


j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4			750	0		
3		2625	0			
2	15750	0				
1	0					

$$B(1,3)=\min\begin{cases}\overset{15750}{B(1,2)}+B(3,3)+\overset{2250}{30.15.5}\\[2ex] B(1,1)+\underset{2625}{B(2,3)}+\underset{5250}{30.35.5}\end{cases}$$

$$B(i,i)=0$$

$$B(i,j)=\min_{k=i}^{j-1} B(i,k)+B(k+1\,,\,j)+R_iC_kC_j$$



i →

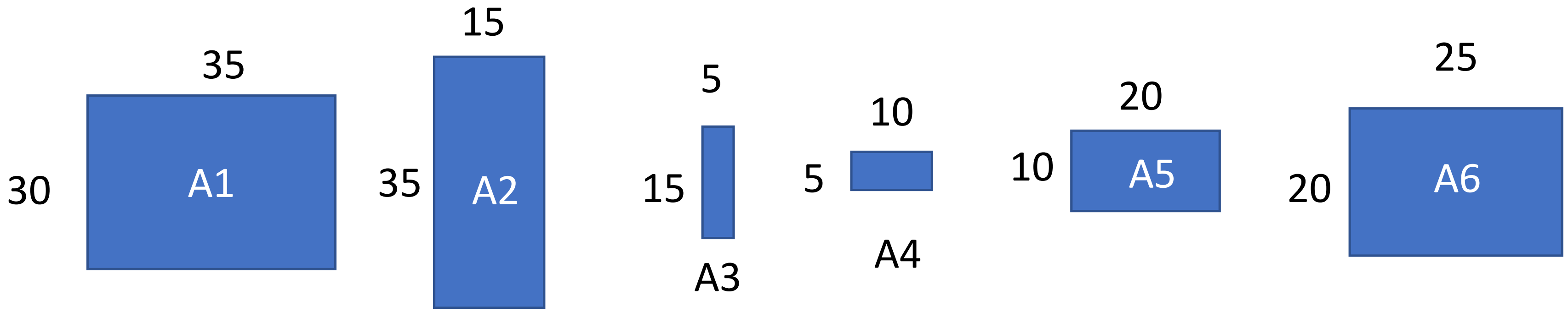
j ↓

j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4			750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$$B(1,3)=\min\begin{cases} \overbrace{B(1,2)}^{15750}+B(3,3)+\overbrace{30.15.5}^{2250} \\ B(1,1)+\underbrace{B(2,3)}_{2625}+\underbrace{30.35.5}_{5250} \end{cases}$$

$$B(i,i)=0$$

$$B(i,j)=\min_{k=i}^{j-1} B(i,k)+B(k+1,j)+R_i C_k C_j$$

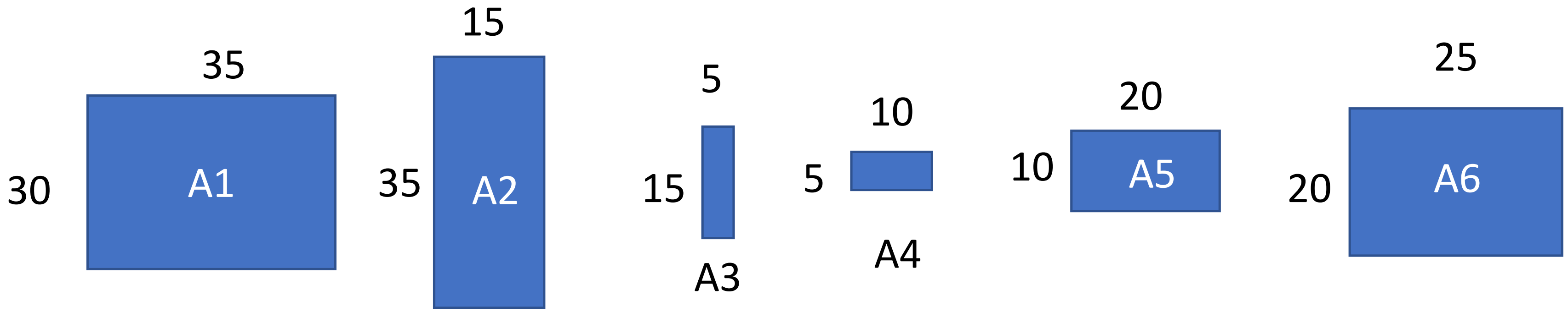


j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4			750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$$B(2,4)=\min\left\{\begin{array}{l} K=3 \\ B(2,3)+B(4,4)+35.5.10 \\ \\ B(2,2)+B(3,4)+35.15.10 \\ K=2 \end{array}\right.$$

$$B(i,i)=0$$

$$B(i,j)=\min_{k=i}^{j-1} B(i,k)+B(k+1,j)+R_iC_kC_j$$

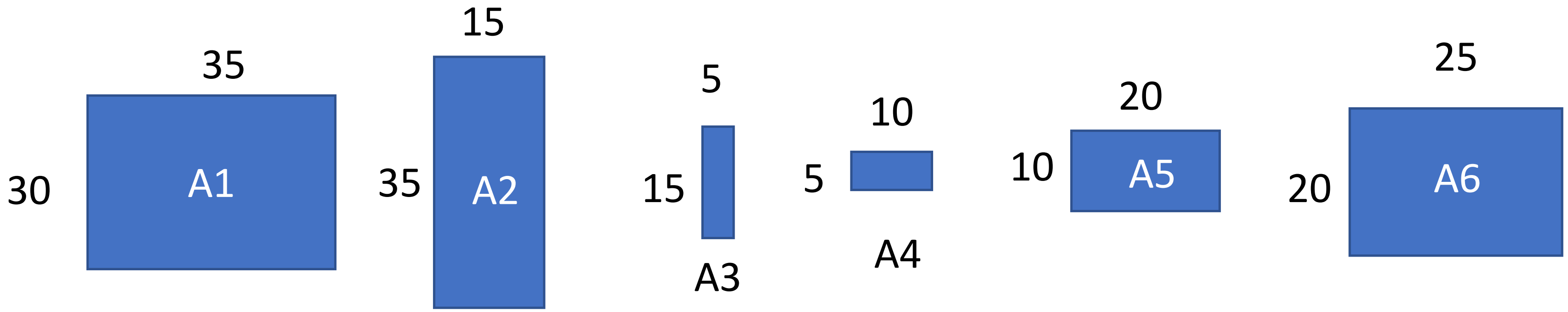


j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4			750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$$B(2,4)=\min\begin{cases} B(2,3)+B(4,4)+35.5.10 \\ B(2,2)+B(3,4)+35.15.10 \end{cases}$$

$$B(i,i)=0$$

$$B(i,j)=\min_{k=i}^{j-1} B(i,k)+B(k+1,j)+R_iC_kC_j$$

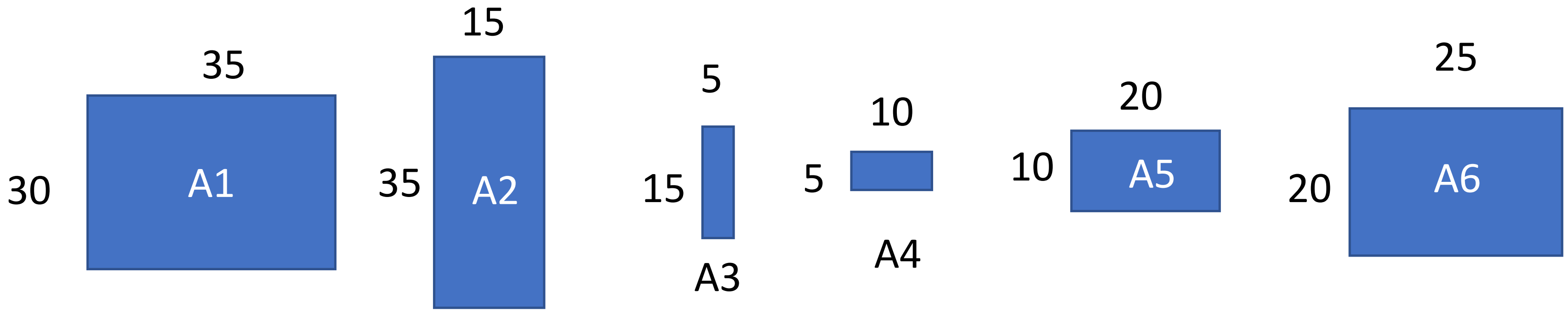


j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4		4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$$B(2,4)=\min\begin{cases} B(2,3)+B(4,4)+35.5.10 \\ B(2,2)+B(3,4)+35.15.10 \end{cases}$$

$$B(i,i)=0$$

$$B(i,j)=\min_{k=i}^{j-1} B(i,k)+B(k+1\,,\,j)+R_iC_kC_j$$



i

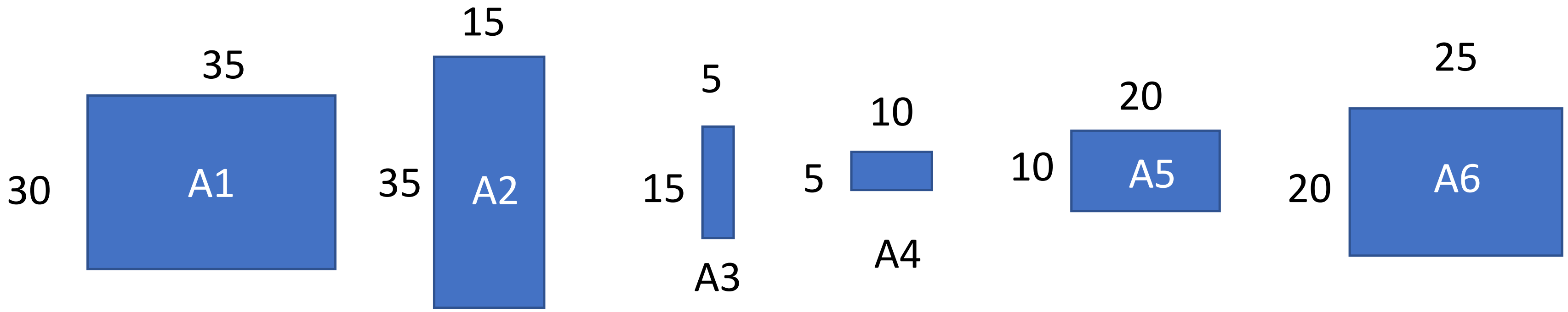
j

j \ i	1	2	3	4	5	6
6					5000	0
5				1000	0	
4		4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$B(3,5)=$

$B(i,i)=0$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$

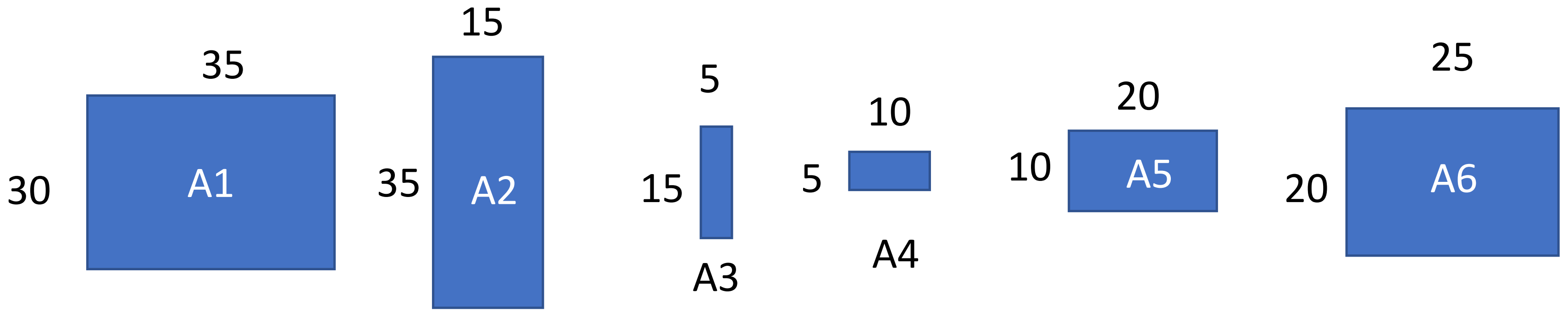


j \ i	1	2	3	4	5	6
6					5000	0
5			2500	1000	0	
4		4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

B(3,5)=

$$B(i,i)=0$$

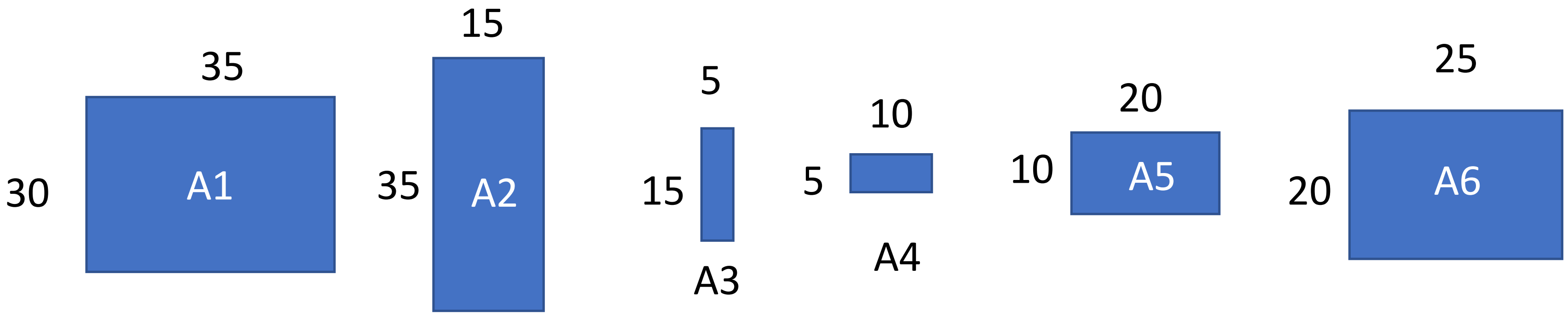
$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\,j) + R_i C_k C_j$$



j \ i	1	2	3	4	5	6
6		10500	5375	3500	5000	0
5	11875	7125	2500	1000	0	
4	9375	4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$$B(i,i)=0$$

$$B(i,j)= \min_{k=i}^{j-1} B(i,k) + B(k+1\,,\, j) + R_i C_k C_j$$

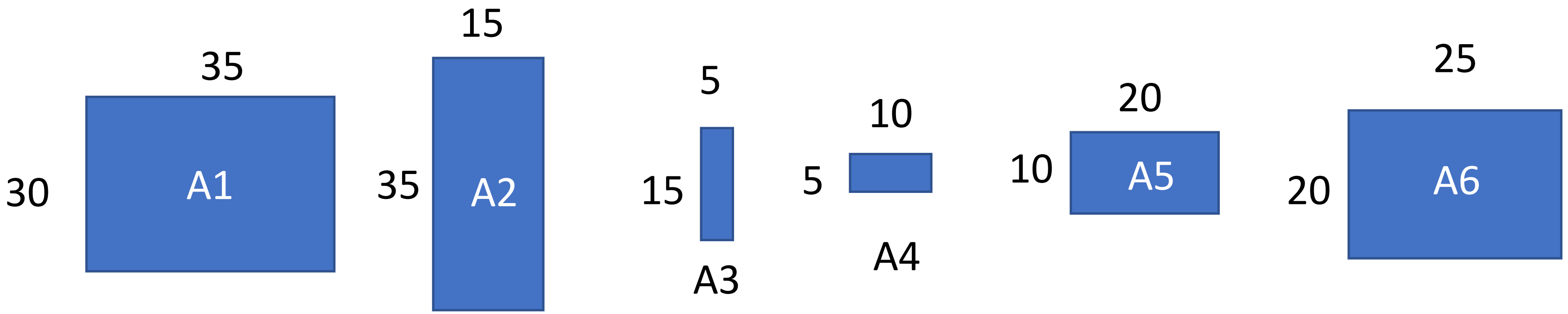


i

j

	1	2	3	4	5	6
6		10500	5375	3500	5000	0
5	11875	7125	2500	1000	0	
4	9375	4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$$B(1,6)=\min\begin{cases} K=1 & B(1,1)+B(2,6)+R1C1C6 \\ K=2 & B(1,2)+B(3,6)+R1C2C6 \\ K=3 & B(1,3)+B(4,6)+R1C3C6 \\ K=4 & B(1,4)+B(5,6)+R1C4C6 \\ K=5 & B(1,5)+B(6,6)+R1C5C6 \end{cases}$$



j \ i	1	2	3	4	5	6
6	15125	10500	5375	3500	5000	0
5	11875	7125	2500	1000	0	
4	9375	4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

$$B(1,6)=\min\begin{cases} K=1 \quad B(1,1)+B(2,6)+R1C1C6 \\ K=2 \quad B(1,2)+B(3,6)+R1C2C6 \\ K=3 \quad B(1,3)+B(4,6)+R1C3C6 \\ K=4 \quad B(1,4)+B(5,6)+R1C4C6 \\ K=5 \quad B(1,5)+B(6,6)+R1C5C6 \end{cases}$$

Matrix Chain Multiplication

Initialize array m[x,y] to zero

Starting at diagonal, working toward upper left $\theta(n^2)$

Compute B[i,j] according to:

B(i,i)=0

$$B(i,j) = \min_{k=i}^{j-1} B(i,k) + B(k+1, j) + R_i C_k C_j \qquad \theta(n)$$

<u>***Runtime:***</u> $\boldsymbol{\theta(n3)}$

Dynamic Programming

lecture 3

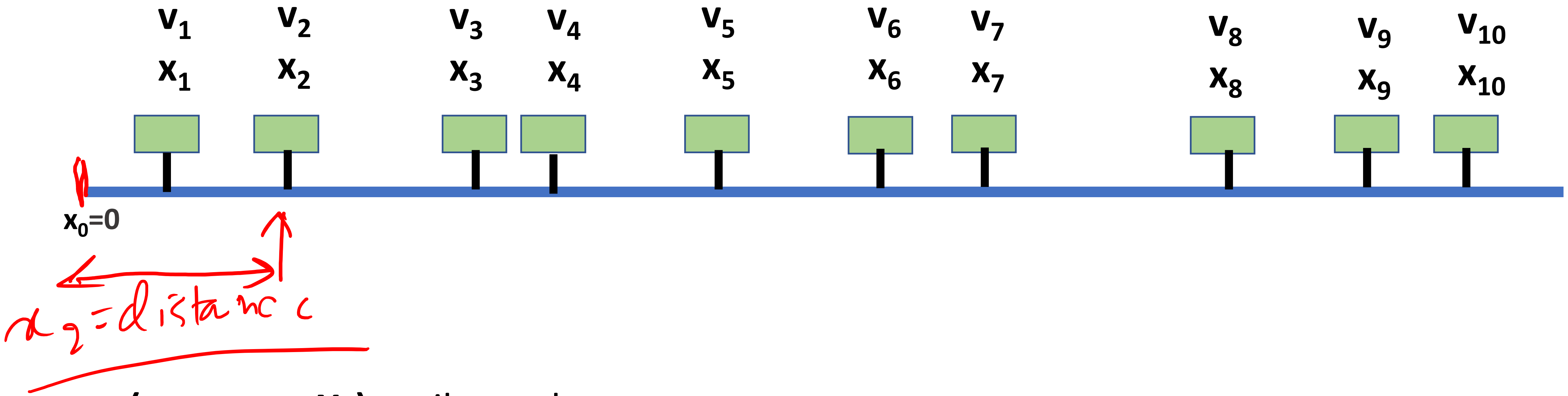

TIME FOR A NEW TRUCK?
Ford
LOWE'S
verizon


Coca-Cola
102° reasons to refresh yourself.
ClearChannel
078706


WINERY
HERMANN · MISSOURI

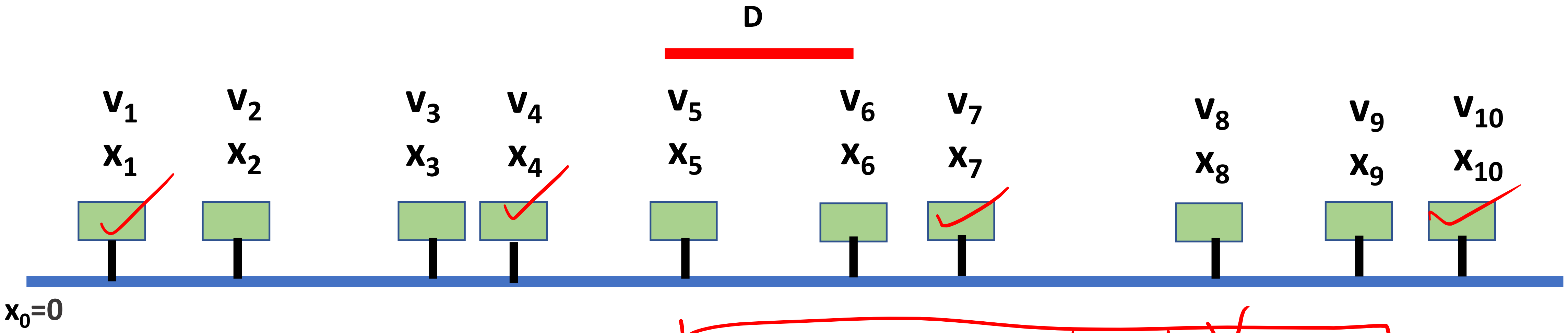

Coca-Cola
97° reasons to refresh yourself.
ClearChannel
078706


Coca-Cola
100° reasons to refresh yourself.
ClearChannel
078706



$(x_1, x_2, x_3, \ldots, X_n)$: mile markers

$(v_1, v_2, v_3, \ldots, v_n)$: Viewership, e.g., v_i = number of people that view billboard at x_i



$\boxed{v_1+v_4+v_7+v_{10}}$ e.g.

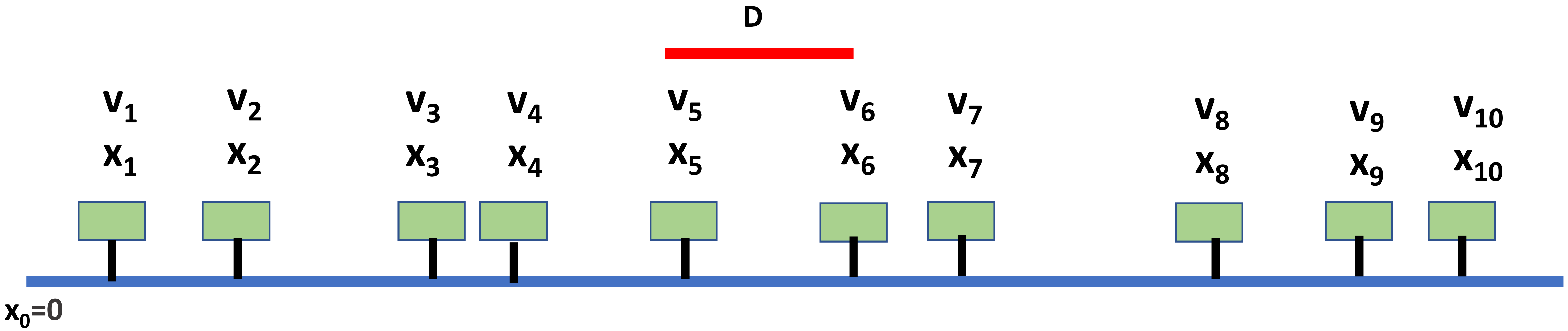
$\boxed{\max}$

$(x_1,x_2,x_3,\dots,X_n)$: mile markers

$(v_1,v_2,v_3,\dots,v_n)$: Viewership, e.g., v_i = number of people that view billboard at x_i

D: distance parameters, can not place ads that are closer than D miles apart

Goal: is to maximize viewership for an acceptable campaign

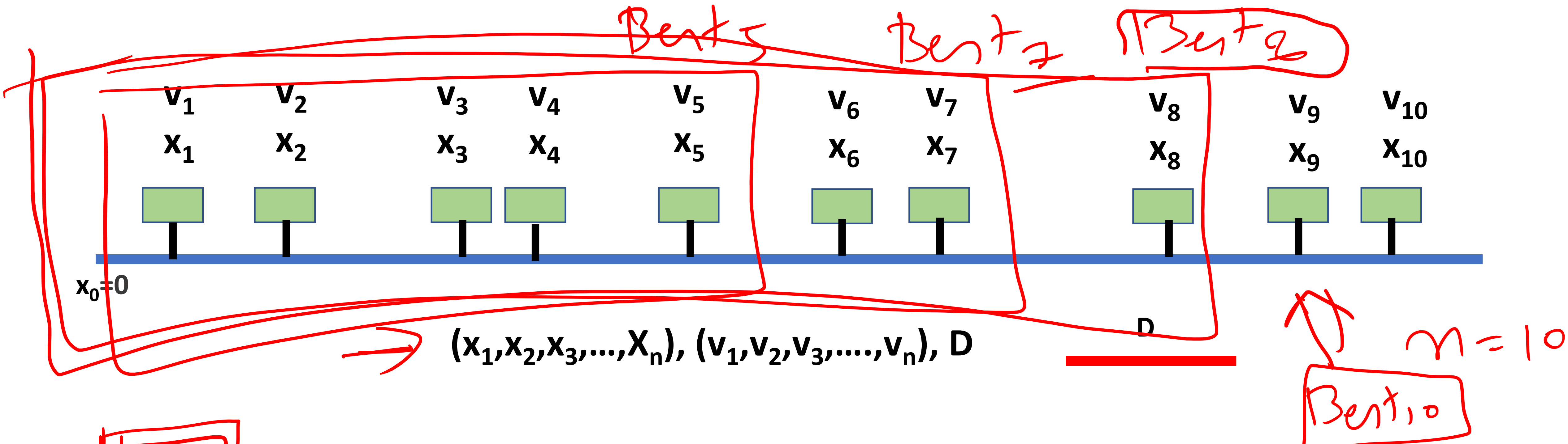


(x_1,x_2,x_3,…,X_n) : mile markers

(v_1,v_2,v_3,….,v_n) : Viewership, e.g., v_i = number of people that view billboard at x_i

D: distance parameters, can not place ads that are closer than D miles apart

Goal: is to maximize viewership for an acceptable campaign



$(x_1,x_2,x_3,\ldots,X_n), (v_1,v_2,v_3,\ldots,v_n), D$

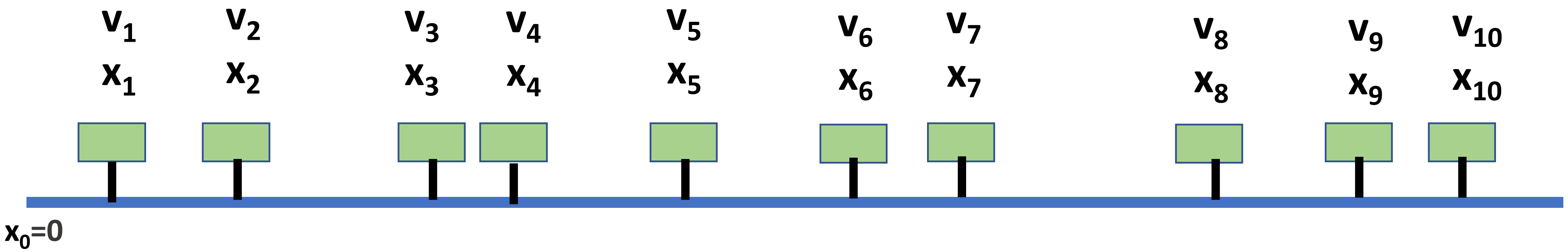
$n=10$

$\boxed{Best_n}$ = max viewership for an acceptable campaign that considers the first n billboards

$Best_j$= max viewership for an acceptable campaign that considers the first j billboards

$j = 1, n$

$j = 7$

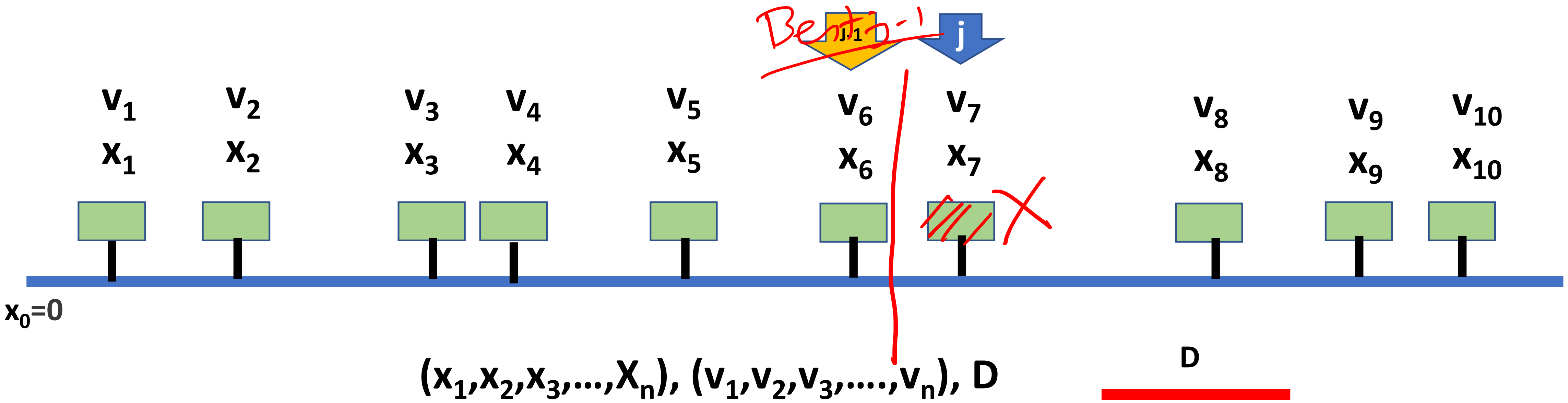


$(x_1,x_2,x_3,\ldots,X_n)$, $(v_1,v_2,v_3,\ldots.,v_n)$, D **D**

$Best_n$ = max viewership for an acceptable campaign that considers the first n billboards

$Best_j$= max viewership for an acceptable campaign that considers the first j billboards

$$Best_j = \max \begin{cases} \mathbf{Best_{j-1}} \\ \mathbf{V_j + Best_{(closest\ billboard\ that\ is\ atleastD\ away)}} \end{cases}$$



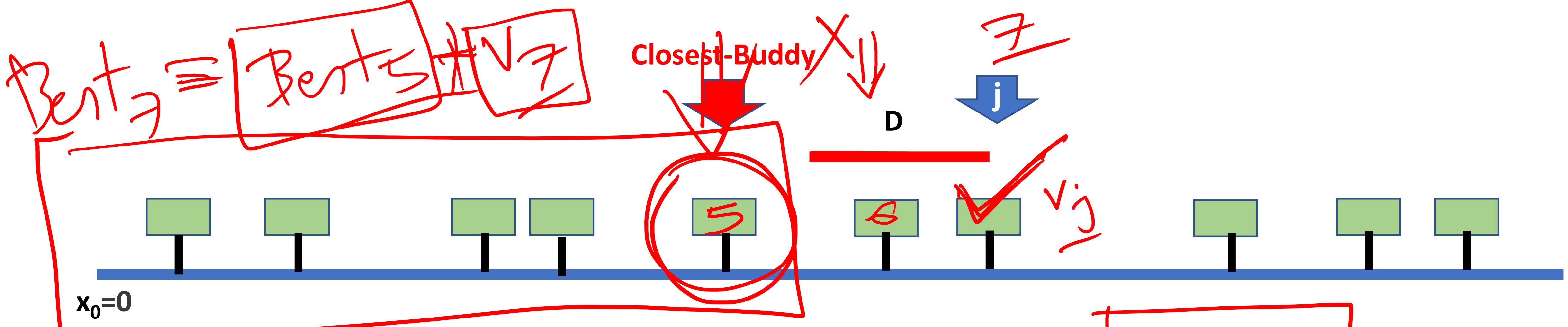
$(x_1,x_2,x_3,\ldots,X_n), (v_1,v_2,v_3,\ldots,v_n)$, D

$Best_n$ = max viewership for an acceptable campaign that considers the first n billboards

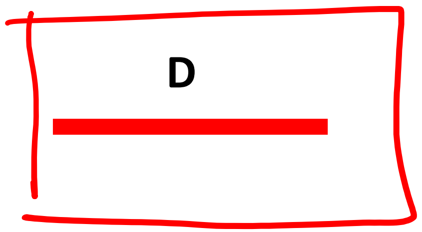
$Best_j$= max viewership for an acceptable campaign that considers the first j billboards

$$Best_j = \max \begin{cases} Best_{j-1} \\ V_j + Best_{\text{(closest billboard that is atleastD away)}} \end{cases}$$

j = 7



$(x_1,x_2,x_3,\ldots,X_n), (v_1,v_2,v_3,\ldots,v_n)$, D



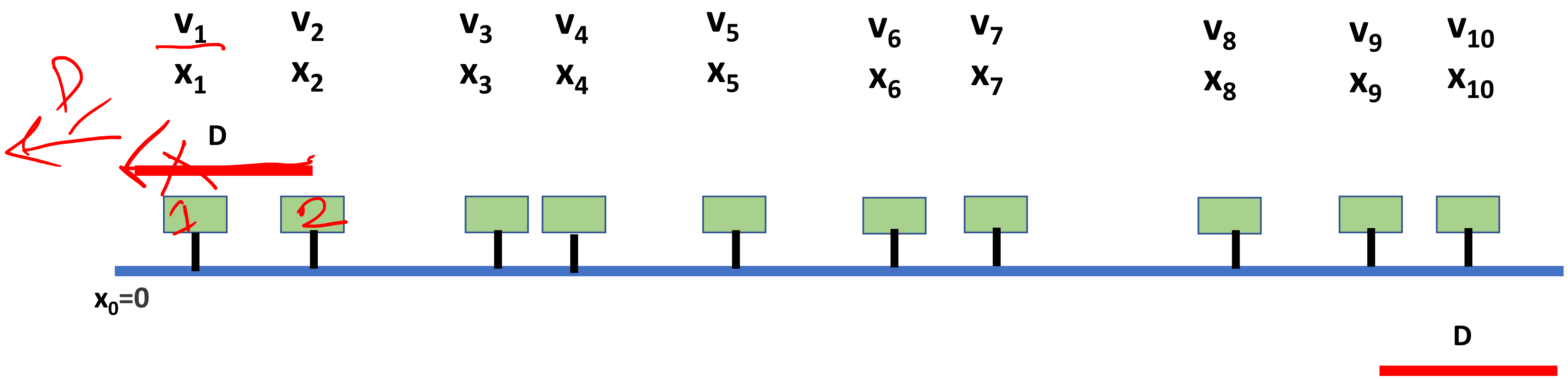
$Best_n$ = max viewership for an acceptable campaign that considers the first n billboards

$Best_j$= max viewership for an acceptable campaign that considers the first j billboards

$$Best_j = \max \begin{cases} \mathbf{Best_{j-1}} \\ V_j + Best_{\text{(closest billboard that is atleastD away)}} \end{cases}$$

Closest-Buddy

j



$Best_1 = v_1$

0

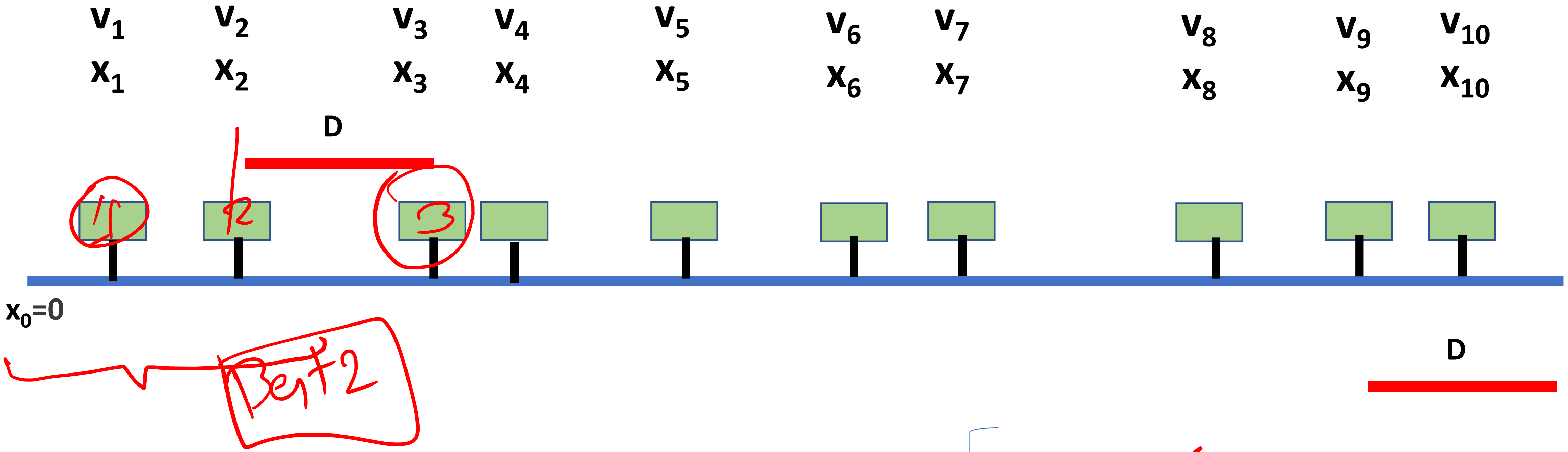
$Best_2$ = Max (Best1 , v2+$Best_{Closest\text{-}Buddy}$)

Max(v_1, v_2)

$$Best_j = \max \begin{cases} Best_{j-1} \\ V_j + Best_{(\text{closest billboard that is atleastD away})} \end{cases}$$

1

$(x_1, x_2, x_3, \ldots, X_n)$, $(v_1, v_2, v_3, \ldots, v_n)$, D



$Best_1 = v_1$

$Best_2 = Max\ (\ Best_1\ ,\ v_2 + Best_{Closest\text{-}Buddy}\ )$

$Best_3 = Max\ (\ Best_2\ ,\ v_3 + Best_{Closest\text{-}Buddy}\ )$
$\qquad = Max(\ v_2\ ,\ v_3 + Best_1\ )$

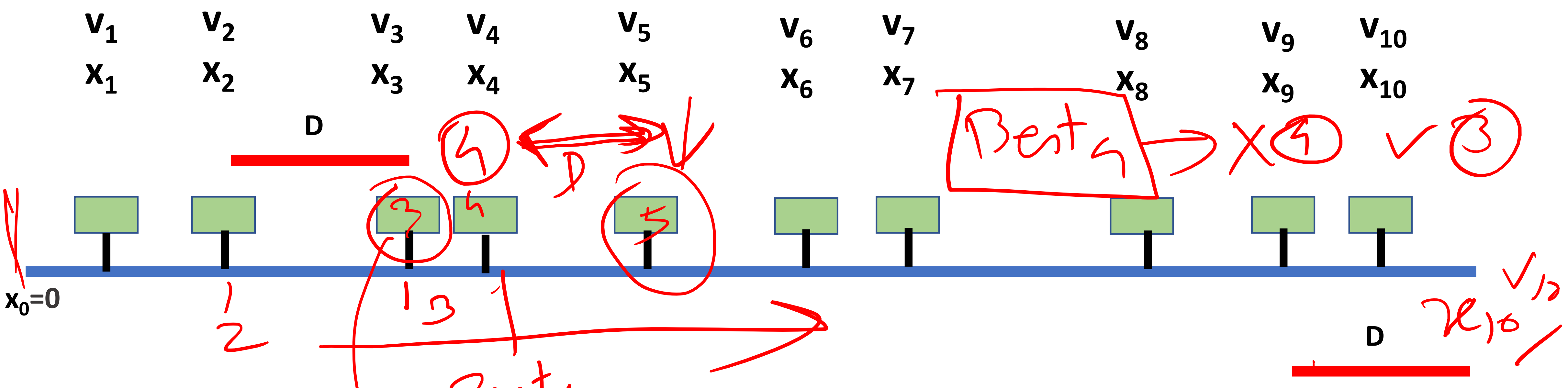
$$Best_j = \max \begin{cases} Best_{j-1} \\ V_j + Best_{(closest\ billboard\ that\ is\ atleastD\ away)} \end{cases}$$

$(x_1, x_2, x_3, \ldots, X_n), (v_1, v_2, v_3, \ldots., v_n), D$

$\boxed{1}$

Best Closest-Buddy for 3

Best 1



$Best_1 = v_1$

$Best_2 = Max\ (\ Best_1\ ,\ v_2 + Best_{Closest\text{-}Buddy}\ )$

$Best_3 = Max\ (\ Best_2\ ,\ v_3 + Best_{Closest\text{-}Buddy}\ )$
$\quad\quad = Max(\ v_2\ ,\ v_3 + Best_1\ )$

$$Best_j = \max \begin{cases} Best_{j-1} \\ V_j + Best_{(\text{closest billboard that is atleastD away})} \end{cases}$$

$(x_1, x_2, x_3, \ldots, X_n), (v_1, v_2, v_3, \ldots., v_n), D$

$$\boxed{(x_3 - x_2) > D}$$

Best

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

Cl(j) := closest buddy that is at least D distance away

j = 5

d = 4

x[5] − x[4] < D

$\theta(n^2)$

i = 6

Best[0]=0

For i=1 to n

cl=i-1

while(dist(x[cl],x[i]<D) cl--;

3 $\theta(n)$

$\theta(n)$

Best[i]=max {best[i-1] , v[i]+best[cl] }

Return best[n]

$\theta(n^2)$

But, we can do better?

d = 3 ?

d − −

d = 2

Pre-Computation to speed up DP Algorithm

Dynamic Programming

1. Has a **recursive solution** to the problem
2. Has **memory**
3. Pick the **correct order** for evaluating the smaller problems

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

Cl(j) := closest buddy that is at least D distance away

```
Best[0]=0
For i=1 to n

    cl=i-1
    while(dist(x[cl],x[i]<D) cl--;

    Best[i]=max {best[i-1] , v[i]+best[cl] }

Return best[n]
```

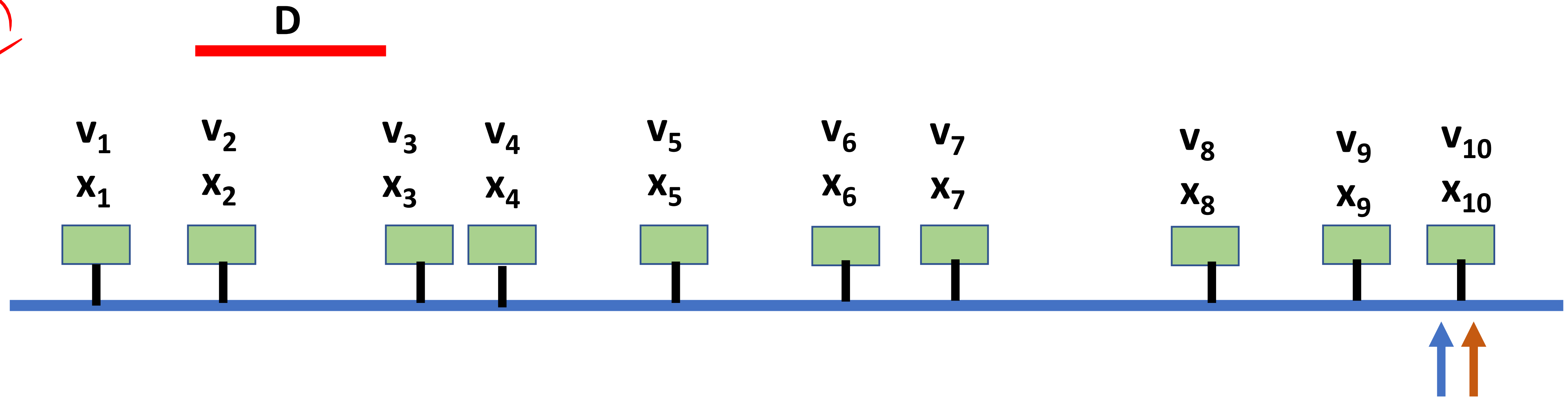
$\theta(n)$

$\boldsymbol{\theta(n^2)}$

$\theta(n)$

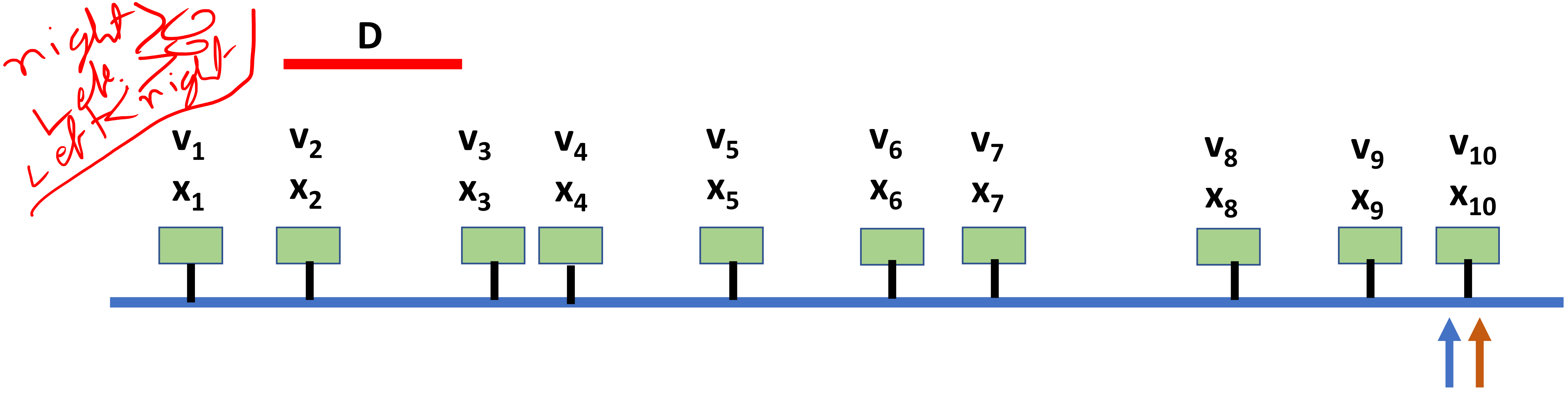
But, we can do better?

$\Theta(n)$



Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

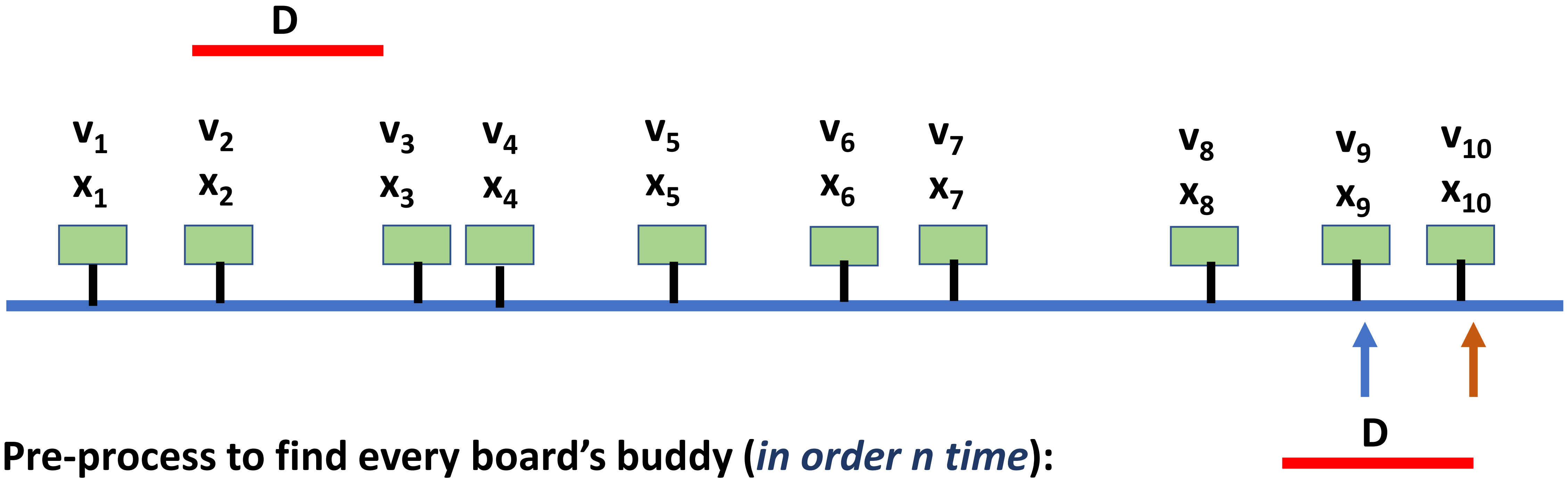


Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

While right and left are valid:

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

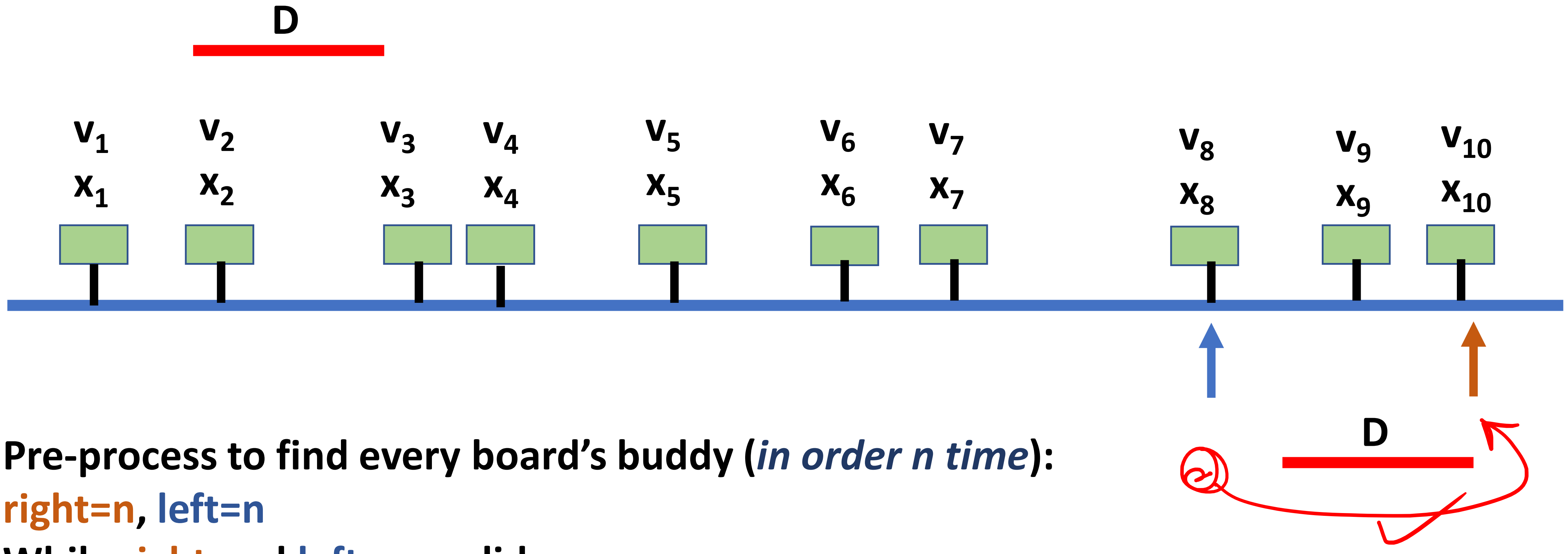


Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

While right and left are valid:

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

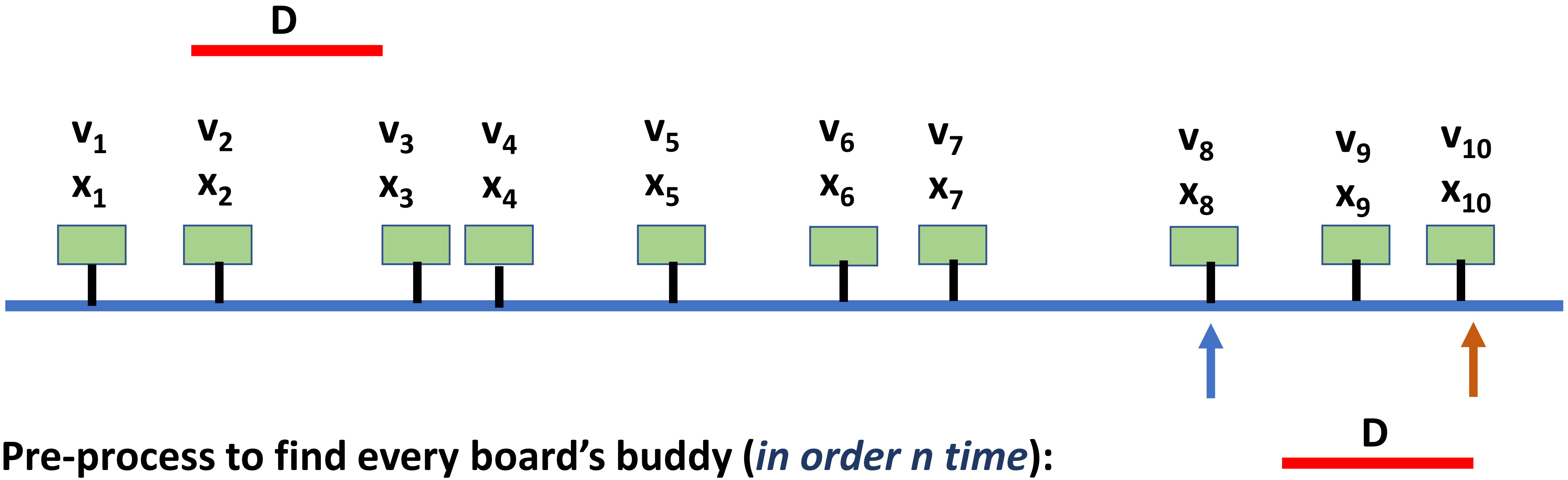


Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

While right and left are valid:

> **move left until distance(x[right],x[left]) > D**
>
> **buddy[right]=left**



Pre-process to find every board's buddy (*in order n time*):

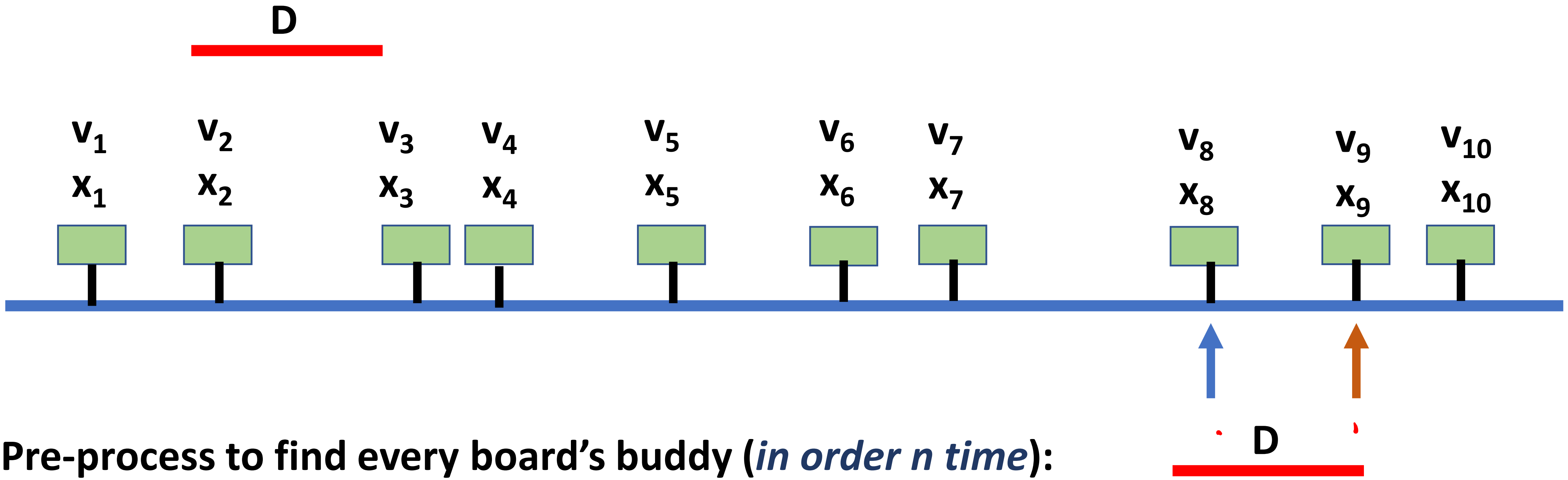
right=n, left=n

While right and left are valid:

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

buddy[right]=left

buddy[10]=8



Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

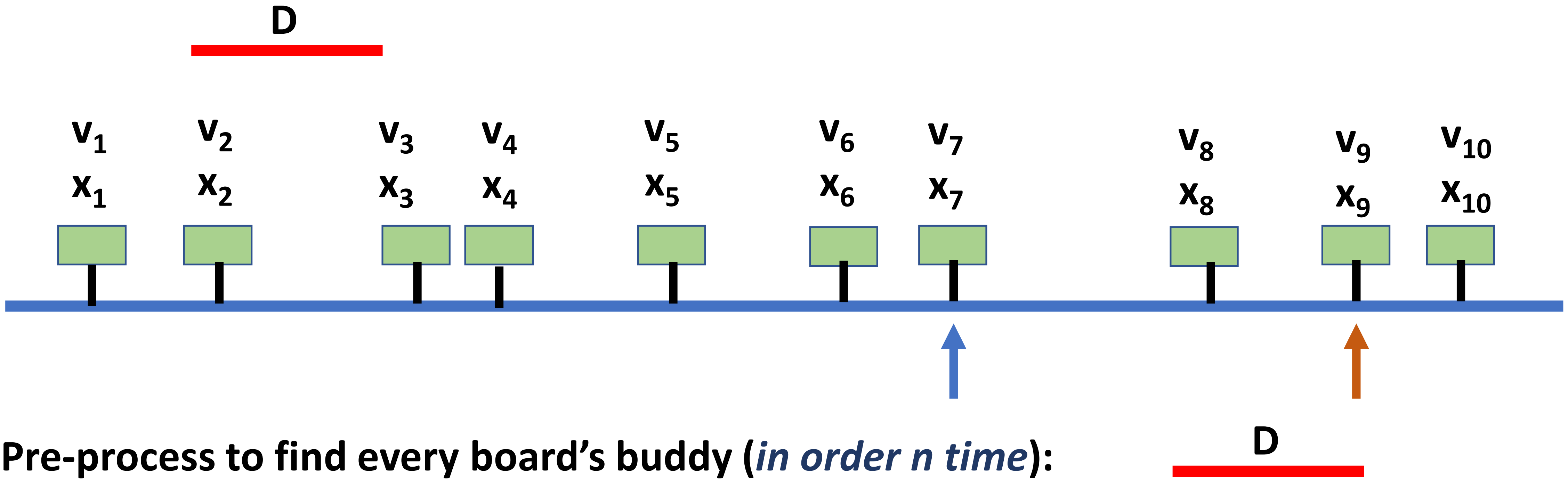
While right and left are valid:

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

  buddy[right]=left

  move right one position

buddy[10]=8



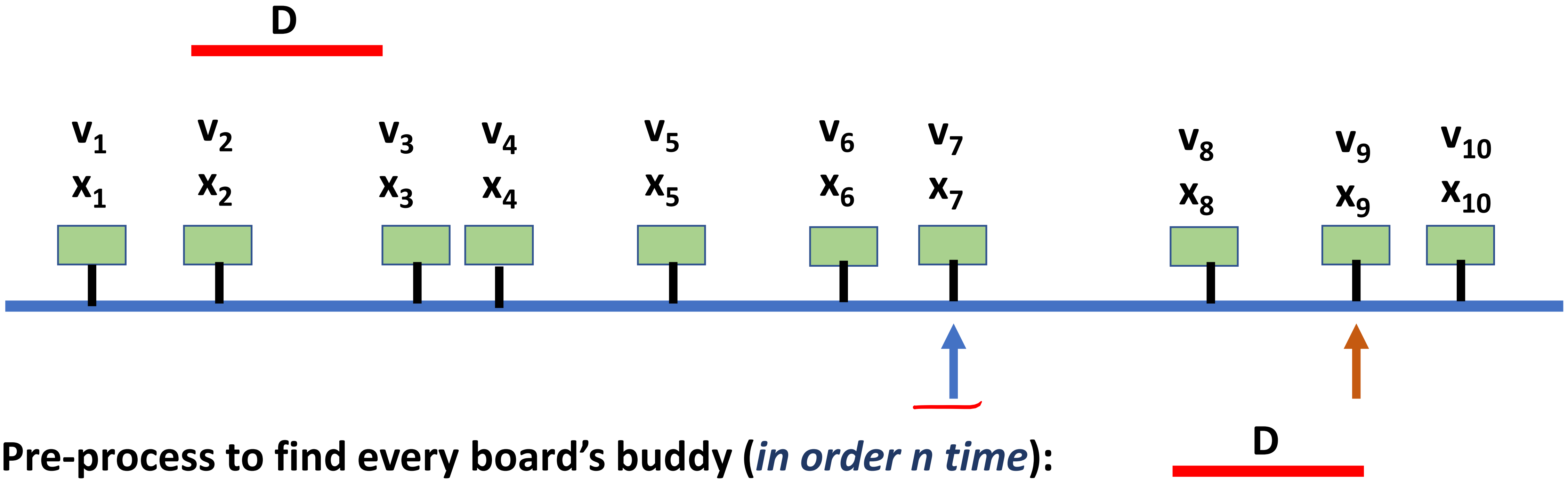
Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

While right and left are valid:

- move left until distance(x[right],x[left]) > D
- buddy[right]=left
- move right one position

buddy[10]=8



Pre-process to find every board's buddy (*in order n time*):

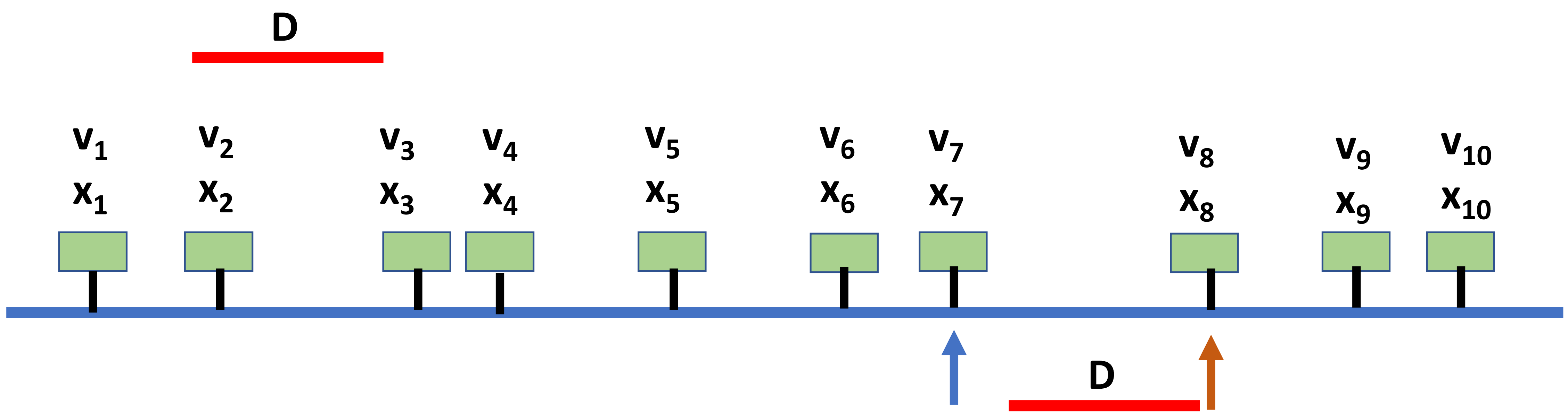
right=n, left=n

While right and left are valid:

> **move left until distance(x[right],x[left]) > D**
>
> **buddy[right]=left**
>
> **move right one position**

buddy[10]=8

buddy[9]=7



Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

While right and left are valid:

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

buddy[right]=left

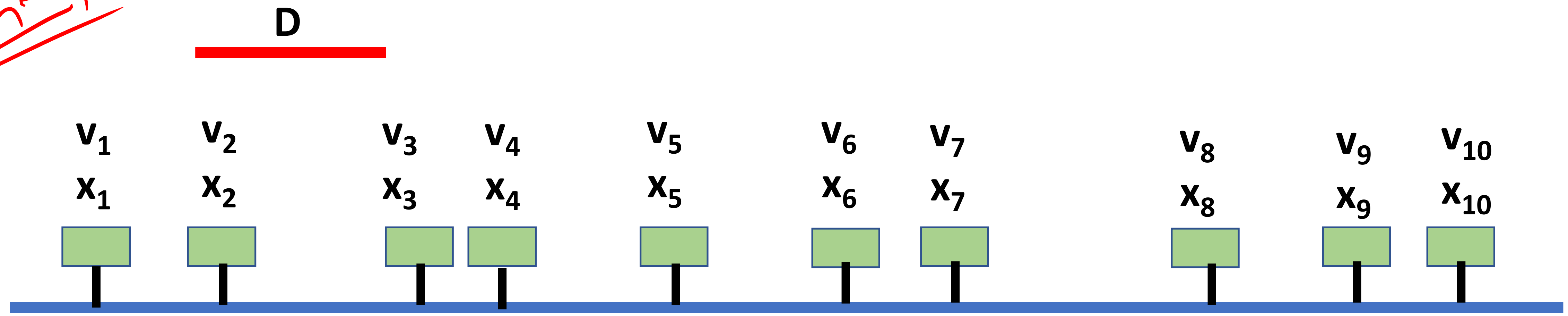
move right one position

buddy[10]=8

buddy[9]=7

buddy[8]=7

$2n-1$



Pre-process to find every board's buddy (*in order n time*):

right=n, left=n

While right and left are valid:

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

buddy[right]=left

move right one position

$O(n)$

buddy[10]=8

buddy[9]=7

buddy[8]=7

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

Cl(j) := closest buddy that is at least D distance away

buddy **:** array

<preprocess buddies> $\theta(n)$

+

Best[0]=0

For i=1 to n $\theta(n)$

left, right

~~cl=i-1~~

~~while(dist(x[cl],x[i]<D) cl--;~~

Best[i]=max {best[i-1] , v[i]+best[**buddy[i]**] } $\boxed{\theta(1)}$

Return best[n]

$$\boxed{\textit{Runtime of the algorithm: } \theta(n)}$$

DNA Testing

- A DNA sequence is a series of nucleotides (ACGT).

One compares DNA for:
- Maternity/paternity testing
- Finding how similar a newly found gene is to existing known genes
- Find what breeds are in your dog through DNA testing

- Finding longest common set of nucleotides.

Longest Common Subsequence

Application: comparison of two Sequence

X={ABCBDAB},Y={BDCABA}

Longest Common Subsequence:

X= A B C D E F G H I J

Y= E C D G I

Which set of common symbols or characters are coming in sequence.

Longest Common Subsequence

Application: comparison of two Sequence

X={ABCBDAB},Y={BDCABA}

Longest Common Subsequence:

X= A B C D E F G H I J

Y= E C D G I

Which set of common symbols or characters are coming in sequence.

Longest Common Subsequence

Application: comparison of two Sequence

X={ABCBDAB},Y={BDCABA}

Longest Common Subsequence:

X= A B C D E F G H I J

Y= E C D G I

Which set of common symbols or characters are coming in sequence.

Longest Common Subsequence

Application: comparison of two Sequence

X={ABCBDAB},Y={BDCABA}

Longest Common Subsequence:

X= A B C D E F G H I J

Y= ~~E~~ C D G I

Which set of common symbols or characters are coming in sequence.

Longest Common Subsequence

n

X= ABCDEFGHIJ

Y= ECDGI

m

Brute-force algorithm

Find our all subsequence of X and Y, and match them.

2^n

Number of subsequence of a n length sequence : $\mathbf{2^n}$

2^m

3

Suppose: {A B C}

Subsequences: {}, {A}, {B}, {C}, {A B}, {A C}, {B C}, {A B C}

Brute-force algorithm run-time $\theta(2^n 2^m)$

Longest Common Subsequence

Define the sub-problems

Base Case:

If one or both strings are {}, then the solution is clearly 0

n

X=ABCB

Y=BDCAB

m

Z={}

X = ABCB

sub X = {}

0

Longest Common Subsequence (LCS)

n

X=ABCB

Y=BDCAB

m

Define the sub-problems

Base Case:

If one or both strings are {}, then the solution is clearly 0

Let LCS (i, j) be the **sub-problem** of LCS(X_n,Y_m) where:

- X_i is the first i characters of string X_n
- Y_j is the first j characters of string Y_m

LCS(2,3)

AB

BDC

Longest Common Subsequence

LCS(4,5)

n

X=ABCB

Y=BDCAB

m

Define the sub-problems

Base Case:

If one or both strings are {}, then the solution is clearly 0

Let LCS (i, j) be the **sub-problem** of LCS(X_n,Y_m) where:

- X_i is the first i characters of string X_n
- Y_j is the first j characters of string Y_m

$\boxed{\text{LCS(4,4)}}$

ABCB

BDCA

Longest Common Subsequence

Key observation

If last two characters match, then we can use LCS of sub-problem and simply add the last two matching characters to it.

If X[i]==Y[j]

$LCS(X_i,Y_j)=LCS(X_{i-1},Y_{j-1}) + 1$

But if last characters do not match?

$LCS(X_{i-1},Y_j) \leftrightarrow LCS(X_i,Y_{j-1})\ )$

X=ABC	X=ABCB
Y=BDC	Y=BD

X=ABCB

Y=BDCAB

X[4]=Y[5]

LCS(3, 4) +1

$LCS(X_4,Y_5)=LCS(X_3,Y_4)+1$

LCS(X, Y)

X=ABCB

Y=BDC

Longest Common Subsequence

LCS(3,4)+1

Max(LCS(3,3), LCS(2,4)

+1

Key observation

X=ABCB

Y=BDCAB

If last two characters match, then we can use LCS of sub-problem and simply add the last two matching characters to it.

X[4]=Y[5]

$LCS(X_4,Y_5)=LCS(X_3,Y_4)+1$

Max

If X[i]==Y[j]

$LCS(X_i,Y_j)=LCS(X_{i-1},Y_{j-1}) + 1$, $LCS(x_{i-1}, y_j)$, $LCS(x_i, y_{j-1})$

But if last characters do not match?

X=ABCB

Y=BDC

$LCS(X_i,Y_j)=MAX(\ LCS(X_{i-1},Y_j)\ ,\ LCS(X_i,Y_{j-1})\ )$

X=ABC	X=ABCB
Y=BDC	Y=BD

Longest Common Subsequence (LCS)

Memory

C[i,j] two-dimensional array that stores LCS(X_i, Y_j)

$$C[i,j] = \begin{cases} C[i-1,j-1] + 1 & \text{if } X[i]=Y[j] \\ \text{Max}(\, C[i-1,j],\ C[i,j-1]\,) & \text{otherwise} \end{cases}$$

Base Case: We start with *i=j=0* (empty substrings of x and y)

If any of the length is 0, X_0, Y_0, their LCS is empty

LCS Example

- We'll see how LCS algorithm works on the following example:
  - X = ABCB
  - Y = BDCAB

3

- What is the Longest Common Subsequence of X and Y?

- LCS(X,Y) = BCB
  - X = A **B** **C** **B**
  - Y = **B** D **C** A **B**

$\Theta(2^4 \cdot 2^5)$

LCS Example (0)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi						
1	**A**						
2	**B**						
3	**C**						
4	**B**						

$\boxed{C}$

X = ABCB; m = |X| = 4
Y = BDCAB; n = |Y| = 5
Allocate array c[5,4]

Longest Common Subsequence

Memory

C[i,j] two-dimensional array that stores LCS(X_i, Y_j)

$$
C[i,j] = \begin{cases} C[i-1,j-1] + 1 & \text{if } X[i]=Y[j] \\ \text{Max}(\, C[i-1,j], C[i,j-1] \,) & \text{otherwise} \end{cases}
$$

Base Case:

We start with $i=j=0$ (empty substrings of x and y)

If any of the length is 0, X_0, Y_0, their LCS is empty length = 0

LCS Example (1)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

for i = 1 to m $\qquad$ c[i,0] = 0
for j = 1 to n $\qquad$ c[0,j] = 0

LCS Example (1)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

for i = 1 to m $\quad\quad$ c[i,0] = 0
for j = 1 to n $\quad\quad$ c[0,j] = 0

LCS Example (2)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**				
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (3)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0		
2	B	0					
3	C	0					
4	B	0					

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (4)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	
2	B	0					
3	C	0					
4	B	0					

i = 1
j = 4

i = 0
j = 3

if (X[i] == Y[j])
  c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

ABCB

BDCAB

LCS Example (5)

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	
2	B	0					
3	C	0					
4	B	0					

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (5)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	→ **1**
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

ABCB

BDCAB

LCS Example (6)

i		j	0	1	2	3	4	5
			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**					
3	**C**		**0**					
4	**B**		**0**					

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

ABCB

BDCAB

LCS Example (6)

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**				
3	**C**	**0**					
4	**B**	**0**					

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

ABCB
BDCAB

LCS Example (7)

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	
3	C	0					
4	B	0					

if (X[i] == Y[j])
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (7)

ABCB
BDCAB

i	Xi \ j	0	1	2	3	4	5
		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	
3	C	0					
4	B	0					

if (X[i] == Y[j])
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (7)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	
3	C	0					
4	B	0					

if (X[i] == Y[j])
        c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (8)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	
3	C		0					
4	B		0					

$i = 2$

$j = 5$

$c[1,4]$

$\boxed{1+1}$

$= 2$

if (X[i] == Y[j])
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (8)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0					
4	B	0					

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (10)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1			
4	B	0					

if (X[i] == Y[j])
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (11)

ABCB
BDCAB

j		0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2		
4	B	0					

if (X[i] == Y[j])
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (12)

ABCB
BDCAB

1, 1, 2, 2, 3

i	j	0	1	2	3	4	5
		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2	2	2
4	B	0					

if (X[i] == Y[j])
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (13)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**				

if (X[i] == Y[j])
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (14)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2	2	2
4	B	0	1	1	2	2	

if (X[i] == Y[j])
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (15)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2	2	2
4	B	0	1	1	2	2	3

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Length Algorithm

$\Theta(n^2)$

LCS-Length(X,Y)

m = length(X) // get the # of symbols in X — $\Theta(1)$

n = length(Y) // get the # of symbols in Y — $\Theta(1)$

for i = 1 to m c[i,0] = 0 // special case: Y_0 — $\Theta(1)$

for j = 1 to n c[0,j] = 0 // special case: X_0 — $\Theta(1)$

for i = 1 to m // for all X_i → $\Theta(n)$

for j = 1 to n // for all Y_j → $\Theta(n)$

if (X[i] == Y[j])

c[i,j] = c[i-1,j-1] + 1

else

c[i,j] = max(c[i-1,j], c[i,j-1]) — $\Theta(1)$

return c[m,n] // return LCS length for X and Y