Dynamic Programming

Outline and Reading

- Matrix Chain-Product (5.3.1)
- Dynamic Programming: The General Technique (5.3.2)
- 0-1 Knapsack Problem (5.3.3)

Matrix Chain Product

Dynamic Programming is a general algorithm design paradigm.

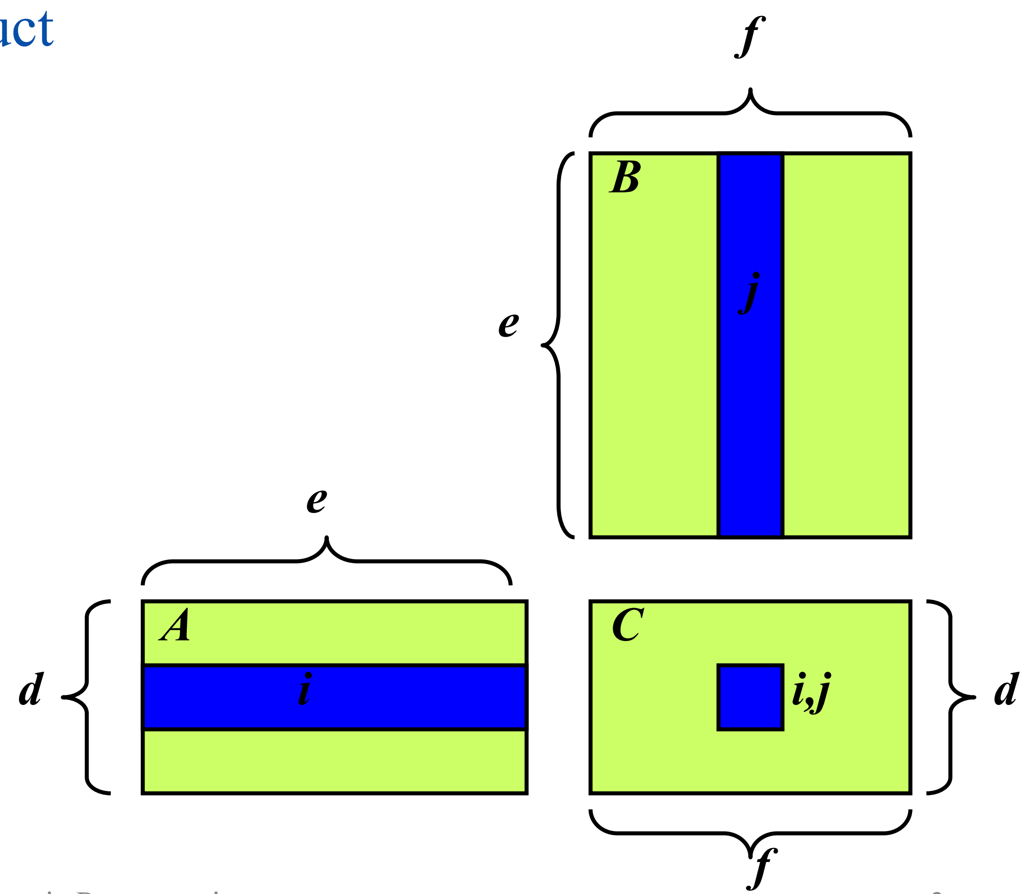
- Rather than give the general structure, we first give a motivating example: Matrix Chain-Product

Review: Matrix Multiplication

- $\boldsymbol{C} = \boldsymbol{A}*\boldsymbol{B}$
- $\boldsymbol{A}$ is $\boldsymbol{d} \times \boldsymbol{e}$ and $\boldsymbol{B}$ is $\boldsymbol{e} \times \boldsymbol{f}$

$$C[i,j] = \sum_{k=0}^{e-1} A[i,k] * B[k,j]$$

- $\boldsymbol{O}(\boldsymbol{d} \cdot \boldsymbol{e} \cdot \boldsymbol{f})$ time



Matrix Chain Product

Matrix Chain-Product**:**

- Compute $A=A_0*A_1*\ldots*A_{n-1}$
- A_i is $d_i \times d_{i+1}$
- **Problem**: How to parenthesize in such a way that **minimizes** the total number of scalar multiplications?

Example:

- B is 3×100
- C is 100×5
- D is 5×5
- (B*C)*D takes 1500 + 75 = 1575 ops
- B*(C*D) takes 1500 + 2500 = 4000 ops

Another Approach: Greedy (v1)

Idea: Repeatedly select the product that uses (up) the most operations.

Counter-example:

- A is 10×5
- B is 5×10
- C is 10×5
- D is 5×10

This greedy approach gives (A*B)*(C*D)

- takes 500+1000+500 = 2000 ops

A better solution: A*((B*C)*D)

- takes 500+250+250 = 1000 ops

Another Approach: Greedy (v2)

Idea: Repeatedly select the product that uses the fewest operations.

Counter-example:

- A is 101×11
- B is 11×9
- C is 9×100
- D is 100×99

This greedy approach gives A*((B*C)*D))

- takes 109989+9900+108900=228789 ops

A better solution is (A*B)*(C*D)

- takes 9999+89991+89100=189090 ops

The greedy approach is not giving us the optimal value.

"Recursive" Approach

Define subproblems:

- Find the best parenthesization of $A_i*A_{i+1}*\dots*A_j$.
- Let $N_{i,j}$ denote the number of operations done by this subproblem.
- The optimal solution for the whole problem is $N_{0,n-1}$.

Subproblem optimality: The optimal solution can be defined in terms of optimal subproblems

- There has to be a final multiplication (root of the expression tree) for the optimal solution.
- Say, the final multiply is at index i: $(A_0*\dots*A_i)*(A_{i+1}*\dots*A_{n-1})$.
- Then the optimal solution $N_{0,n-1}$ is the sum of two optimal subproblems, $N_{0,i}$ and $N_{i+1,n-1}$ plus the time for the last multiply.
- If the global optimum did not have these optimal subproblems, we could define an even better "optimal" solution.

Characterizing Equation

- The global optimal has to be defined in terms of optimal subproblems, depending on where the final multiply is at.

- Consider all possible places for that final multiply:
  - Recall that A_i is a $d_i \times d_{i+1}$ dimensional matrix.
  - So, a characterizing equation for $N_{i,j}$ is the following:

$$N_{i,j} = \min_{i \le k < j} \{ N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1} \}$$

- Note that subproblems are not independent – meaning subproblems overlap.

Dynamic Programming Algorithm Visualization

$$N_{i,j} = \min_{i \le k < j} \{N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}\}$$

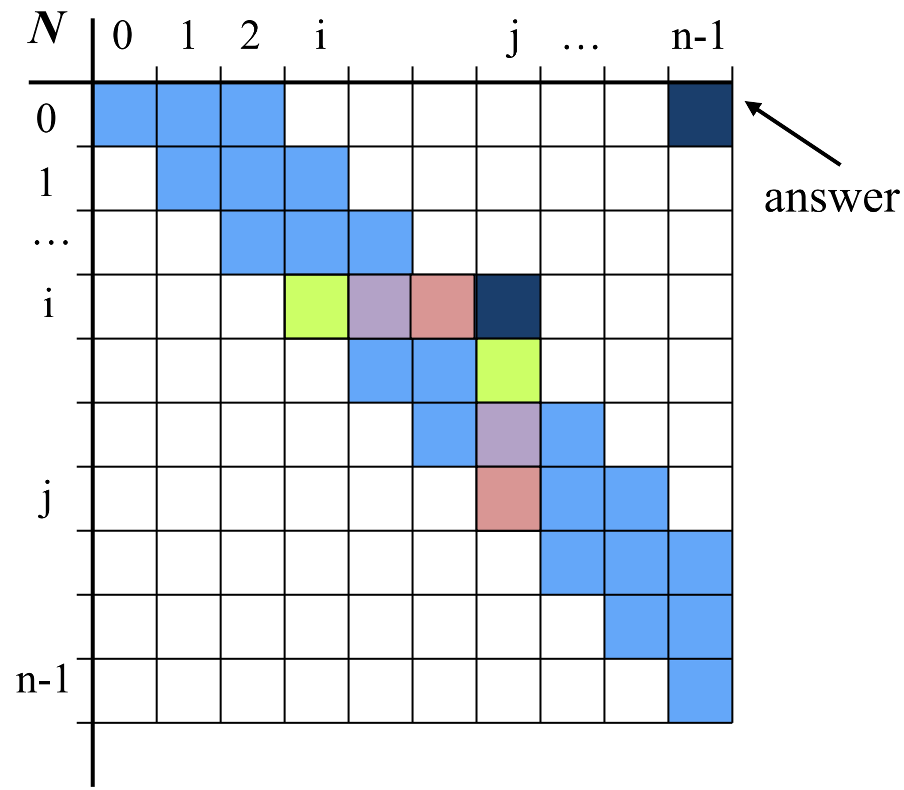
The bottom-up construction fills in the N array by diagonals

$N_{i,j}$ gets values from previous entries in i-th row and j-th column

Filling in each entry in the N table takes O(n) time.

- Total run time: $O(n^3)$

Getting actual parenthesization can be done by remembering "k" for each N entry in a separate table



Dynamic Programming Algorithm

Since subproblems overlap, we don't use recursion.

Instead, we construct optimal subproblems "bottom-up."

$N_{i,i}$'s are easy, so start with them

Then do problems of "length" 2,3,… subproblems, and so on.

Running time: $O(n^3)$

Algorithm ***matrixChain***(***S***):

Input: sequence ***S*** of ***n*** matrices to be multiplied

Output: number of operations in an optimal parenthesization of ***S***

for $i \leftarrow 0$ **to** $n - 1$ **do**

$N_{i,i} \leftarrow \mathbf{0}$

for $length \leftarrow 1$ **to** $n - 1$ **do**

{ $length = j - i$ is the length of the chain }

for $i \leftarrow 0$ **to** $n - 1 - length$ **do**

$j \leftarrow i + length$

$N_{i,j} \leftarrow +\infty$

for $k \leftarrow i$ **to** $j - 1$ **do**

$N_{i,j} \leftarrow \min\{N_{i,j}, N_{i,k} + N_{k+1,j} + d_i\, d_{k+1}\, d_{j+1}\}$

record k that produces minimum $N_{i,j}$

return $N_{0,n-1}$

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

N	0	1	2	3	4	5
0	0					
1		0				
2			0			
3				0		
4					0	
5						0

number of scalar operations required to multiply

$$A_i \cdot A_{i+1} \cdot A_{i+2} \cdot \ldots \cdot A_{j-1} \cdot A_j$$

k	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

matrix index where final multiplication occurred to obtain optimal solution given in N[i][j]

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

N	0	1	2	3	4	5
0	0	15750				
1		0				
2			0			
3				0		
4					0	
5						0

k	0	1	2	3	4	5
0		0				
1						
2						
3						
4						
5						

number of scalar operations required to multiply

$$A_i \cdot A_{i+1} \cdot A_{i+2} \cdot \ldots \cdot A_{j-1} \cdot A_j$$

matrix index where final multiplication occurred to obtain optimal solution given in N[i][j]

$A_0 \cdot A_1$ N[0][1] = 0 + 0 + 30*35*15 = 15750

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

$d_0 \quad d_1 \quad d_2 \quad d_3 \quad d_4 \quad d_5 \quad d_6$

end j

start i

N	0	1	2	3	4	5
0	0	15750				
1		0	2625			
2			0			
3				0		
4					0	
5						0

number of scalar operations required to multiply

$$A_i \cdot A_{i+1} \cdot A_{i+2} \cdot \ldots \cdot A_{j-1} \cdot A_j$$

k	0	1	2	3	4	5
0		0				
1			1			
2						
3						
4						
5						

matrix index where final multiplication occurred to obtain optimal solution given in N[i][j]

$A_0 \cdot A_1$ N[0][1] = 0 + 0 + 30*35*15 = 15750

$A_1 \cdot A_2$ N[1][2] = 0 + 0 + 35*15*5 = 2626

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

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

k	0	1	2	3	4	5
0		0				
1			1			
2				2		
3						
4						
5						

number of scalar operations required to multiply

$$A_i \cdot A_{i+1} \cdot A_{i+2} \cdot \ldots \cdot A_{j-1} \cdot A_j$$

matrix index where final multiplication occurred to obtain optimal solution given in N[i][j]

$A_0 \cdot A_1$ N[0][1] = 0 + 0 + 30*35*15 = 15750

$A_1 \cdot A_2$ N[1][2] = 0 + 0 + 35*15*5 = 2626

$A_2 \cdot A_3$ N[2][3] = 0 + 0 + 15*5*10 = 750

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

$d_0 \quad d_1 \quad d_2 \quad d_3 \quad d_4 \quad d_5 \quad d_6$

end j

start i

N	0	1	2	3	4	5
0	0	15750				
1		0	2625			
2			0	750		
3				0	1000	
4					0	
5						0

number of scalar operations required to multiply

$$A_i \cdot A_{i+1} \cdot A_{i+2} \cdot \ldots \cdot A_{j-1} \cdot A_j$$

k	0	1	2	3	4	5
0		0				
1			1			
2				2		
3					3	
4						
5						

matrix index where final multiplication occurred to obtain optimal solution given in N[i][j]

$A_0 \cdot A_1$ N[0][1] = 0 + 0 + 30*35*15 = 15750

$A_1 \cdot A_2$ N[1][2] = 0 + 0 + 35*15*5 = 2626

$A_2 \cdot A_3$ N[2][3] = 0 + 0 + 15*5*10 = 750

$A_3 \cdot A_4$ N[3][4] = 0 + 0 + 5*10*20 = 1000

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

N	0	1	2	3	4	5
0	0	15750				
1		0	2625			
2			0	750		
3				0	1000	
4					0	5000
5						0

number of scalar operations required to multiply

$$A_i \cdot A_{i+1} \cdot A_{i+2} \cdot \ldots \cdot A_{j-1} \cdot A_j$$

k	0	1	2	3	4	5
0		0				
1			1			
2				2		
3					3	
4						4
5						

matrix index where final multiplication occurred to obtain optimal solution given in N[i][j]

$A_0 \cdot A_1$ N[0][1] = 0 + 0 + 30*35*15 = 15750

$A_1 \cdot A_2$ N[1][2] = 0 + 0 + 35*15*5 = 2626

$A_2 \cdot A_3$ N[2][3] = 0 + 0 + 15*5*10 = 750

$A_3 \cdot A_4$ N[3][4] = 0 + 0 + 5*10*20 = 1000

$A_4 \cdot A_5$ N[4][5] = 0 + 0 + 10*20*25 = 5000

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

N	0	1	2	3	4	5
0	0	15750	7875			
1		0	2625			
2			0	750		
3				0	1000	
4					0	5000
5						0

k	0	1	2	3	4	5
0		0	0			
1			1			
2				2		
3					3	
4						4
5						

$(A_0)\cdot(A_1\cdot A_2)$ = 0 + 2625 + 30*35*5 = 7875

$(A_0\cdot A_1)\cdot(A_2)$ = 15750 + 0 + 30*15*5 = 18000

matrix:		A_0		A_1		A_2		A_3		A_4		A_5
dimension:		30x35		35x15		15x5		5x10		10x20		20x25
	d_0		d_1		d_2		d_3		d_4		d_5		d_6

end j

start i

N	0	1	2	3	4	5
0	0	15750	7875			
1		0	2625	4375		
2			0	750		
3				0	1000	
4					0	5000
5						0

k	0	1	2	3	4	5
0		0	0			
1			1	2		
2				2		
3					3	
4						4
5						

$$(A_1)\cdot(A_2\cdot A_3) \quad = 0 + 750 + 35*15*10 = 6000$$

$$(A_1\cdot A_2)\cdot(A_3) \quad = 2625 + 0 + 35*5*10 = 4375$$

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

N	0	1	2	3	4	5
0	0	15750	7875			
1		0	2625	4375		
2			0	750	2500	
3				0	1000	
4					0	5000
5						0

k	0	1	2	3	4	5
0		0	0			
1			1	2		
2				2	2	
3					3	
4						4
5						

$(A_2) \cdot (A_3 \cdot A_4)$ = 0 + 1000 + 15*5*20 = 2500

$(A_2 \cdot A_3) \cdot (A_4)$ = 750 + 0 + 15*10*20 = 3750

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

N	0	1	2	3	4	5
0	0	15750	7875			
1		0	2625	4375		
2			0	750	2500	
3				0	1000	3500
4					0	5000
5						0

k	0	1	2	3	4	5
0		0	0			
1			1	2		
2				2	2	
3					3	4
4						4
5						

$(A_3) \cdot (A_4 \cdot A_5)$ = 0 + 5000 + 5*10*25 = 6250

$(A_3 \cdot A_4) \cdot (A_5)$ = 1000 + 0 + 5*20*25 = 3500

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

$d_0 \quad d_1 \quad d_2 \quad d_3 \quad d_4 \quad d_5 \quad d_6$

end j

start i

N	0	1	2	3	4	5
0	0	15750	7875	9375		
1		0	2625	4375		
2			0	750	2500	
3				0	1000	3500
4					0	5000
5						0

k	0	1	2	3	4	5
0		0	0	2		
1			1	2		
2				2	2	
3					3	4
4						4
5						

$(A_0)\cdot(A_1\cdot A_2\cdot A_3)$ = 0 + 4375 + 30*35*10 = 14875

$(A_0\cdot A_1)\cdot(A_2\cdot A_3)$ = 15750 + 750 + 30*15*10 = 21000

$(A_0\cdot A_1\cdot A_2)\cdot(A_3)$ = 7875 + 0 + 30*5*10 = 9375

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

N	0	1	2	3	4	5
0	0	15750	7875	9375		
1		0	2625	4375	7125	
2			0	750	2500	
3				0	1000	3500
4					0	5000
5						0

k	0	1	2	3	4	5
0		0	0	2		
1			1	2	2	
2				2	2	
3					3	4
4						4
5						

$$(A_1)\cdot(A_2\cdot A_3\cdot A_4) = 0 + 2500 + 35*15*20 = 13000$$

$$(A_1\cdot A_2)\cdot(A_3\cdot A_4) = 2625 + 1000 + 35*5*20 = 7125$$

$$(A_1\cdot A_2\cdot A_3)\cdot(A_4) = 4375 + 0 + 35*10*20 = 11375$$

matrix:		A_0		A_1		A_2		A_3		A_4		A_5	
dimension:		30x35		35x15		15x5		5x10		10x20		20x25	
	d_0		d_1		d_2		d_3		d_4		d_5		d_6

end j

start i

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

k	0	1	2	3	4	5
0		0	0	2		
1			1	2	2	
2				2	2	2
3					3	4
4						4
5						

$(A_2)\cdot(A_3\cdot A_4\cdot A_5)$ = 0 + 3500 + 15*5*25 = 5375

$(A_2\cdot A_3)\cdot(A_4\cdot A_5)$ = 750 + 5000 + 15*10*25 = 9500

$(A_2\cdot A_3\cdot A_4)\cdot(A_5)$ = 2500 + 0 + 15*20*25 = 10000

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

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

k	0	1	2	3	4	5
0		0	0	2	2	
1			1	2	2	
2				2	2	2
3					3	4
4						4
5						

$(A_0) \cdot (A_1 \cdot A_2 \cdot A_3 \cdot A_4)$ = 0 + 7125 + 30*35*20 = 28125

$(A_0 \cdot A_1) \cdot (A_2 \cdot A_3 \cdot A_4)$ = 15750 + 2500 + 30*15*20 = 27250

$(A_0 \cdot A_1 \cdot A_2) \cdot (A_3 \cdot A_4)$ = 7875 + 1000 + 30*5*20 = 11875

$(A_0 \cdot A_1 \cdot A_2 \cdot A_3) \cdot (A_4)$ = 9375 + 0 + 30*10*20 = 15375

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

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

k	0	1	2	3	4	5
0		0	0	2	2	
1			1	2	2	2
2				2	2	2
3					3	4
4						4
5						

$(A_1) \cdot (A_2 \cdot A_3 \cdot A_4 \cdot A_5)$ = 0 + 5375 + 35*15*25 = 18500

$(A_1 \cdot A_2) \cdot (A_3 \cdot A_4 \cdot A_5)$ = 2625 + 3500 + 35*5*25 = 10500

$(A_1 \cdot A_2 \cdot A_3) \cdot (A_4 \cdot A_5)$ = 4375 + 5000 + 35*10*25 = 18125

$(A_1 \cdot A_2 \cdot A_3 \cdot A_4) \cdot (A_5)$ = 7125 + 0 + 35*20*25 = 24625

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

d_0 d_1 d_2 d_3 d_4 d_5 d_6

end j

start i

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

k	0	1	2	3	4	5
0		0	0	2	2	2
1			1	2	2	2
2				2	2	2
3					3	4
4						4
5						

$$(A_0)\cdot(A_1\cdot A_2\cdot A_3\cdot A_4\cdot A_5) = 0 + 10500 + 30*35*25 = 36750$$

$$(A_0\cdot A_1)\cdot(A_2\cdot A_3\cdot A_4\cdot A_5) = 15750 + 5375 + 30*15*25 = 32375$$

$$(A_0\cdot A_1\cdot A_2)\cdot(A_3\cdot A_4\cdot A_5) = 7875 + 3500 + 30*5*25 = 15125$$

$$(A_0\cdot A_1\cdot A_2\cdot A_3)\cdot(A_4\cdot A_5) = 9375 + 5000 + 30*10*25 = 21875$$

$$(A_0\cdot A_1\cdot A_2\cdot A_3\cdot A_4)\cdot(A_5) = 11875 + 0 + 30*20*25 = 26875$$

matrix:	A_0	A_1	A_2	A_3	A_4	A_5
dimension:	30x35	35x15	15x5	5x10	10x20	20x25

$d_0 \quad d_1 \quad d_2 \quad d_3 \quad d_4 \quad d_5 \quad d_6$

end j

start i

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

k	0	1	2	3	4	5
0		0	0	2	2	2
1			1	2	2	2
2				2	2	2
3					3	4
4						4
5						

optimal order in which the following matrices should be multiplied:

$$[(A_0) \cdot (A_1 \cdot A_2)] \cdot [(A_3 \cdot A_4) \cdot (A_5)]$$

General Dynamic Programming Technique

Applies to an optimization problem that at first seems to require a lot of time (possibly exponential), provided we have:

- Simple subproblems: the subproblems can be defined in terms of a few variables, such as *j*, *k*, *l*, *m*, and so on.

- Subproblem overlap: the subproblems are not independent, but instead they overlap (hence, should be constructed bottom-up).

- Subproblem optimality: the global optimum value can be defined in terms of optimal subproblems

0/1 Knapsack Problem

Given: A set S of n items, with each item i having

- w_i - a positive weight
- b_i - a positive benefit

Goal: Choose items with maximum total benefit but with weight at most W.

If we are not allowed to take fractional amounts, then this is the 0/1 knapsack problem.

- In this case, we let T denote the set of items we take
- Objective: maximize

$$\sum_{i \in T} b_i$$

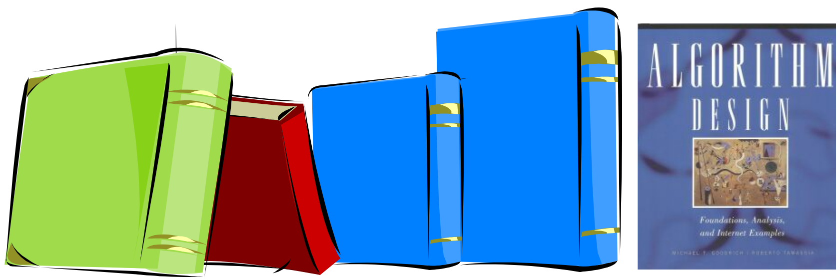
- Constraint:

$$\sum_{i \in T} w_i \leq W$$

Example

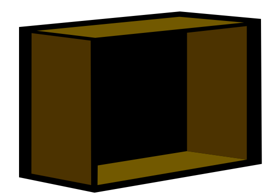
- Given: A set S of n items, with each item i having
  - b_i - a positive "benefit"
  - w_i - a positive "weight"
- Goal: Choose items with maximum total benefit but with weight at most W.

Items:



	1	**2**	**3**	**4**	**5**
Weight:	4 in	2 in	2 in	6 in	2 in
Benefit:	$20	$3	$6	$25	$80

"knapsack"



box of width 9 in

Solution:

- item 5 ($80, 2 in)
- item 3 ($6, 2in)
- item 1 ($20, 4in)

A 0/1 Knapsack Algorithm: First Attempt

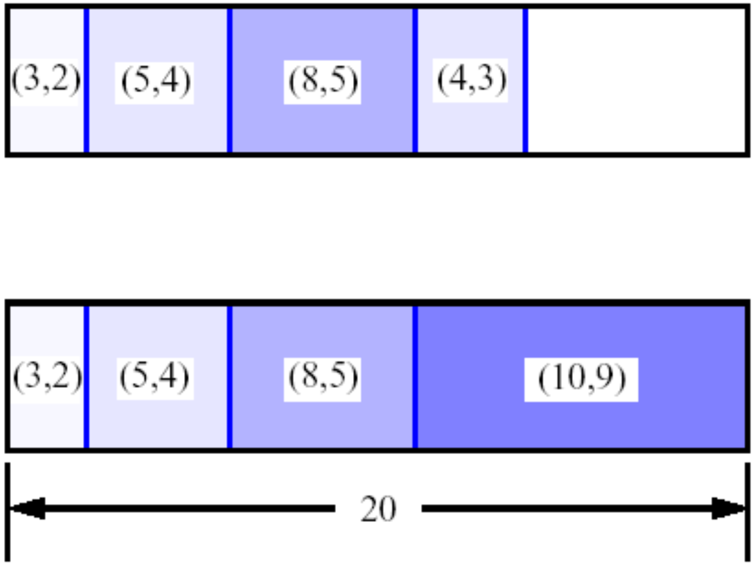
S_k: Set of items numbered 1 to k.

- Idea: Define B[k] = best selection from S_k.

- Problem: does not have subproblem optimality.
  - Consider set S={(3,2),(5,4),(8,5),(4,3),(10,9)} of (benefit, weight) pairs and total weight W = 20

Best for S_4:

Best for S_5:



A 0/1 Knapsack Algorithm: Second Attempt

S_k: Set of items numbered 1 to k.

- <u>Idea</u>: Define B[k,w] to be the best selection from S_k with weight at most w
- Good news: this does have subproblem optimality.

$$B[k,w] = \begin{cases} B[k-1,w] & \text{if } w_k > w \\ \max\{B[k-1,w], B[k-1,w-w_k]+b_k\} & \text{else} \end{cases}$$

That is, the best subset of S_k with weight at most w is either

- the best subset of S_{k-1} with weight at most w or
- the best subset of S_{k-1} with weight at most $w-w_k$ plus item k

0/1 Knapsack Algorithm

$$B[k,w] = \begin{cases} B[k-1,w] & \text{if } w_k > w \\ \max\{B[k-1,w],\, B[k-1,w-w_k]+b_k\} & \text{else} \end{cases}$$

- Recall the definition of B[k,w]
- Since B[k,w] is defined in terms of B[k–1,*], we can use two arrays of instead of a matrix
- Running time: O(nW).
- Not a polynomial-time algorithm since W may be large
- This is a pseudo-polynomial time algorithm

Algorithm ***01Knapsack(S, W)*:**
    Input: set S of n items with benefit b_i and weight w_i; maximum weight W
    Output: benefit of best subset of S with weight at most W
    let A and B be arrays of length $W + 1$
    for $w \leftarrow 0$ **to** W **do**
        $B[w] \leftarrow 0$
    for $k \leftarrow 1$ **to** n **do**
        copy array B into array A
        for $w \leftarrow w_k$ **to** W **do**
            if $A[w-w_k] + b_k > A[w]$ **then**
                $B[w] \leftarrow A[w-w_k] + b_k$
    return $B[W]$