Lecture 16: Dynamic Programming over Intervals

DP Over Intervals

Main idea of interval dynamic programming.

- Problem contains items 1,2,…n. Recurrence gives optimal solution of the original problem[1,n] as function of optimal solution of subproblems of smaller length (length refers to the number of items in the problem)
  - Base case contains problems of length 1, i.e., problem[1,1], problem [2,2],…, problem [n,n].
    - All solutions are stored in diagonals of a 2D array. Solutions of small problems are re-used by bigger ones.
  - Then, we solve n-1 problems of length 2, i.e., problem[1,2], problem[2,3],…, problem[n-1,n].
  - ….
  - Then, we solve n-l+1 problems of length l, i.e., problem[1, l], problem[2,l+1],…, problem [n-l+1,n]
  - Finally, we solve 1 problem of size n: problem [1,n]
- Algorithm fills diagonals in a 2D table from smallest to largest problem length. First the main diagonal (base case), then the one on top of that,….At the end, the solution of the problem[1,n] is at the top right cell.

Longest Palindromic Substring

Def: A palindrome is a string that reads the same backward or forward.

Ex:

- **`radar, level, racecar, madam`**
- "A man, a plan, a canal – Panama!" (ignoring space, punctuation, etc.)

Problem: Given a string $X = x_1 x_2 \dots x_n$, find the longest palindromic substring.

Ex:

- $X =$ **`ACCABA`**
- Palindromic substrings: **`CC, ACCA, ABA`**
- Longest palindromic substring: **`ACCA`**

Note:

- Brute-force algorithm takes $O(n^3)$ time.
- Recall: A substring must be contiguous

Dynamic Programming Solution

Def: Let $p[i,j]$ be $true$ iff $X[i..j]$ is a palindrome.

The Recurrence:

Initial Conditions (subproblems of sizes 1 & 2)

- $p[i,i] = true$, for all i
  - **ACBBCABA**

- $p[i,i+1] = true$ if $x_i = x_{i+1}$
  - **ACBBCABA**

The Actual Recurrence

- $p[i,j] = true$

  if $x_i = x_j$ AND $p[i+1,j-1] = true$
  - **ACBBCABA**
  - **ACBBCABA**

i	1	2	3	4	5	6	7	8
	B	A	B	B	C	C	C	B

A Completed DP Table

Initial Condition
j=i; j=i+1

i/j	1	2	3	4	5	6	7	8
1	T	F						
2		T	F					
3			T	T				
4				T	F			
5					T	T		
6						T	T	
7							T	F
8								T

j=i+2

i/j	1	2	3	4	5	6	7	8
1	T	F	T					
2		T	F	F				
3			T	T	F			
4				T	F	F		
5					T	T	T	
6						T	T	F
7							T	F
8								T

j=i+3

i/j	1	2	3	4	5	6	7	8
1	T	F	T	F				
2		T	F	F	F			
3			T	T	F	F		
4				T	F	F	F	
5					T	T	T	F
6						T	T	F
7							T	F
8								T

j=i+4

i/j	1	2	3	4	5	6	7	8
1	T	F	T	F	F			
2		T	F	F	F	F		
3			T	T	F	F	F	
4				T	F	F	F	T
5					T	T	T	F
6						T	T	F
7							T	F
8								T

j>i+4

i/j	1	2	3	4	5	6	7	8
1	T	F	T	F	F	F	F	F
2		T	F	F	F	F	F	F
3			T	T	F	F	F	F
4				T	F	F	F	T
5					T	T	T	F
6						T	T	F
7							T	F
8								T

Largest is BCCCB

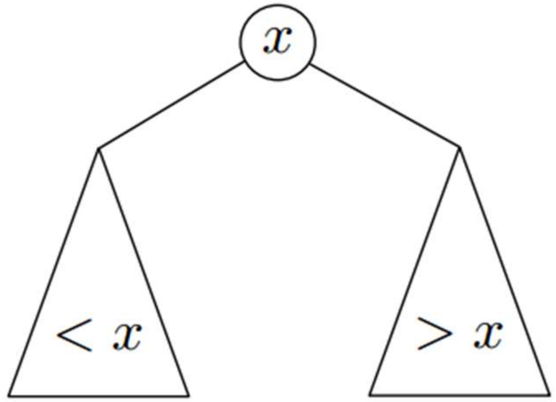
The Algorithm

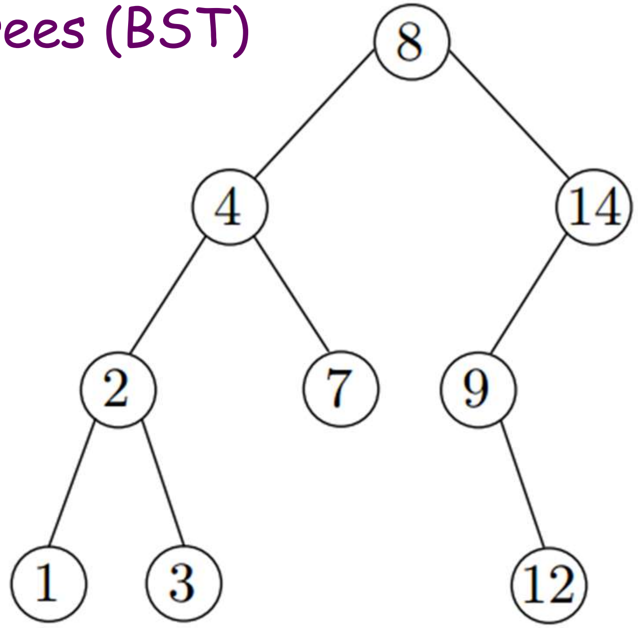
```
max ← 1
for i ← 1 to n − 1 do                         initial conditions
    p[i, i] ← true                                   length 1
    if x_i = x_{i+1} then                            length 2
        p[i, i + 1] ← true,  max ← 2
    else p[i, i + 1] ← false
for l ← 3 to n do                        for each length 3 to n
    for i ← 1 to n − l + 1 do            starting character
        j ← i + l − 1                     ending character
        if p[i + 1, j − 1] = true and x_i = x_j then
            p[i, j] ← true,  max ← l
        else p[i, j] ← false
return max
```

Running time: $O(n^2)$

Space: $O(n^2)$ but can be improved to $O(n)$

Binary search trees (BST)





Tree-Search(T,k):

$x \leftarrow T.root$

while $x \neq nil$ **and** $k \neq x.key$ **do**

  if $k < x.key$ **then** $x \leftarrow x.left$

  else $x \leftarrow x.right$

return x

The (worst-case) search time in a balanced BST is $\Theta(\log n)$

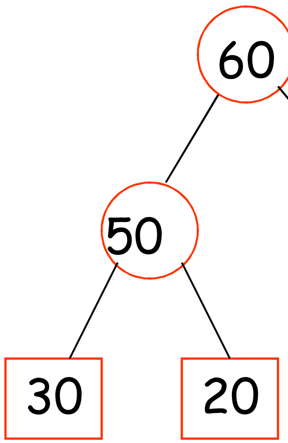
Q: If we know the probability of each key being searched for, can we design a (possibly unbalanced) BST to optimize the expected search time?

Similar Problem seen in Huffman Codes: Binary Merge Tree

You are given a set of leaf nodes $a_1, \dots, a_n$ and associated leaf weights $w(a_1), \dots, w(a_n)$

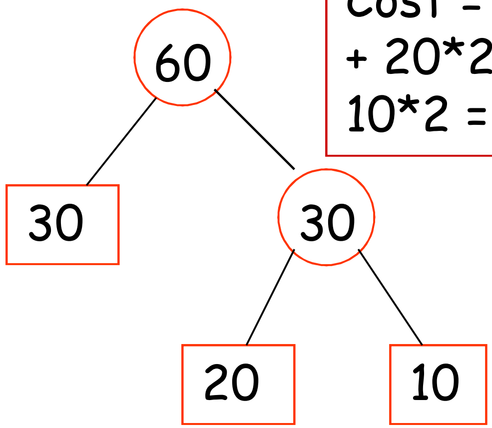
Create a binary tree from the leaf nodes towards the root, in which the size of each node is the sum of the sizes of the two children.

A binary merge tree is **optimal** if it minimizes the weighted external path length. The *weighted external path length* of the tree is $B(T) = \sum_{i=1}^{n} w(a_i) d(a_i)$



Cost = 30*2 + 20*2 + 10*1 = 110

Merge L_1 and L_2, then with L_3



Cost = 30*1 + 20*2 + 10*2 = 90

Merge L_2 and L_3, then with L_1

Optimal Binary Merge Tree Greedy Algorithm - Same as Huffman Coding

Input: $n \geq 2$ leaf nodes, each with a size (i.e., # list elements) .

Output: a binary tree with the given leaf nodes which has a minimum total weighted external path lengths

Algorithm:

Create a min-heap $T[1..n]$ based on the n initial sizes.

While (the heap size ≥ 2) do

extract from the heap two smallest values a and b

create intermediate node of size $a + b$ whose children are a and b

insert the value $(a + b)$ into the heap

Time complexity $O(n\log n)$

It can be shown that the Binary Merge Tree is optimal

The Optimal Binary Search Tree Problem

Problem Definition (simpler than the version in textbook):
Given n keys $a_1 < a_2 < \cdots < a_n$, with weights $f(a_1), \dots, f(a_n)$, find a binary search tree T on these n keys such that

$$B(T) = \sum_{i=1}^{n} f(a_i)(d(a_i) + 1)$$

is minimized, where $d(a_i)$ is the depth of a_i.

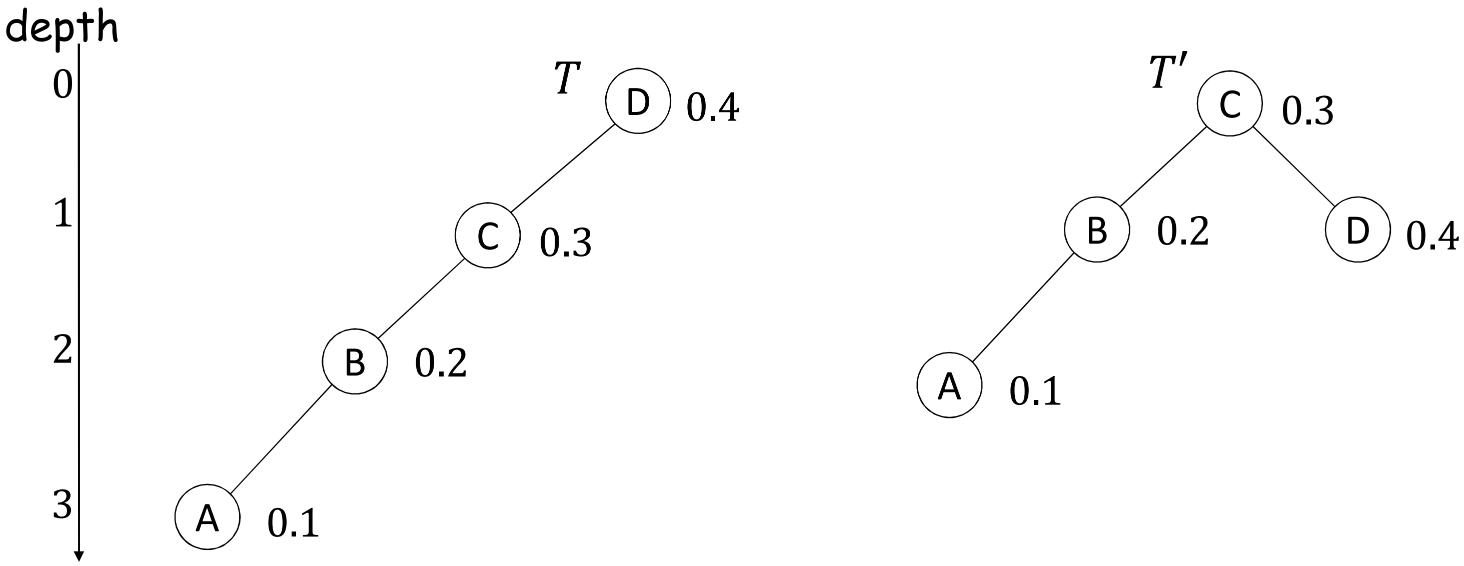
Note: Similar to the Binary Merge Tree problem but with 2 key differences:

- The tree has to be a BST, i.e., the keys are stored in sorted order. In a Binary Merge Tree, there is no ordering among the leaves.
- Keys appear as both internal and leaf nodes. In a Binary Merge Tree, keys (characters) appear only at the leaf nodes.

Motivation: If the weights are the probabilities of the elements being searched for, such a BST will minimize the expected search cost.

Greedy Won't Work

Cannot apply Huffman algorithm because it assumes that all keys must be at leaves. Alternative greedy algorithm: Always pick the heaviest key as root, then recursively build the tree top-down.



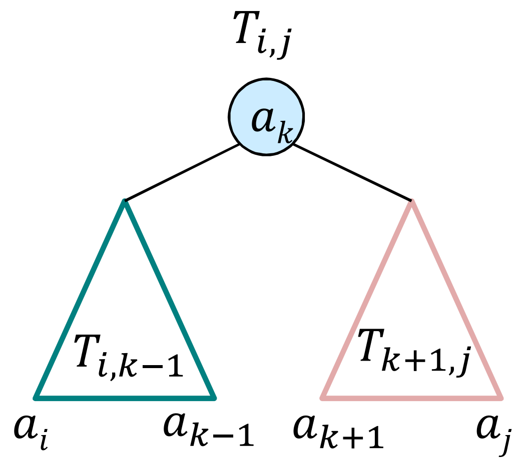
T was built using greedy strategy and has cost $B(T) = 0.4 \cdot 1 + 0.3 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 = 2$

T' has a smaller cost $B(T') = 0.4 \cdot 2 + 0.3 \cdot 1 + 0.2 \cdot 2 + 0.1 \cdot 3 = 1.8$

Dynamic Programming: The Recurrence

Let $T_{i,j}$ be some tree on the subset of nodes $a_i < a_{i+1} < \cdots < a_j$.
The cost is well defined as $B(T_{i,j}) = \sum_{t=i}^{j} f(a_t)(d(a_t) + 1)$

Let $w[i,j] = f(a_i) + \cdots + f(a_j)$



Suppose we ***knew*** root of $T_{i,j}$ was a_k.

$T_{i,j}$ is a BST, so left and right sub-tree children of a_k
are some tree $T_{i,k-1}$ on $a_i < \cdots < a_{k-1}$
and some tree $T_{k+1,j}$ on $a_{k+1} < \cdots < a_j$

Nodes in $T_{i,k-1}$ and $T_{k+1,j}$ are one level deeper in $T_{i,j}$ than in their original trees. So the cost of $T_{i,j}$ *is*

$$
\begin{aligned}
B(T_{i,j}) &= (B(T_{i,k-1}) + w[i,k-1]) \;+\; f(a_k) \;+\; (B(T_{k+1,j}) + w[k+1,j]) \\
&= B(T_{i,k-1}) + B(T_{k+1,j}) + \quad w[i,k-1] + f(a_k) + w[k+1,j] \\
&= B(T_{i,k-1}) + B(T_{k+1,j}) + \quad w[i,j]
\end{aligned}
$$

Dynamic Programming: The Recurrence

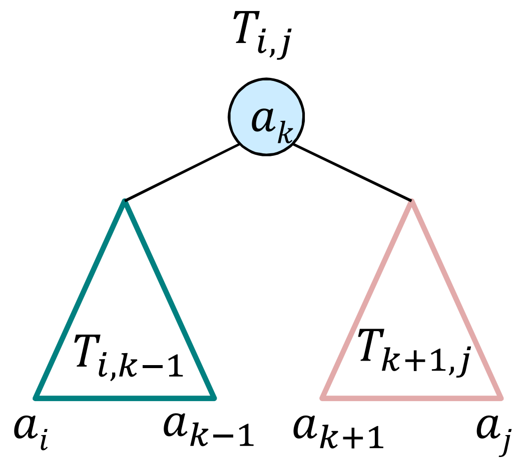
Let $T_{i,j}$ be some tree on the subset of nodes $a_i < a_{i+1} < \cdots < a_j$.
The cost is well defined as $B(T_{i,j}) = \sum_{t=i}^{j} f(a_t)(d(a_t)+1)$

Let $w[i,j] = f(a_i) + \cdots + f(a_j)$

Suppose we ***knew*** root of $T_{i,j}$ was a_k.

The cost of $T_{i,j}$ is

(*) $B(T_{i,j}) = B(T_{i,k-1}) + B(T_{k+1,j}) + w[i,j]$.



In particular, suppose $T_{i,j}$ has root a_k and is a minimum cost tree over all trees with nodes $a_i, \dots, a_j$
$\Rightarrow$ its left subtree $T_{i,k-1}$ must be a minimum cost tree with nodes $a_i, \dots, a_{k-1}$

If it wasn't, we could replace $T_{i,k-1}$ by a lesser cost subtree with nodes $a_i, \dots, a_{k-1}$.
By (*), this would reduce $B(T_{i,j})$, contradicting that $T_{i,j}$ is a minimum cost tree.

Similarly, the right subtree $T_{k+1,j}$ must be minimum cost for $a_{k+1}, \dots, a_j$

Dynamic Programming: The Recurrence

Def: $e[i,j]$ = the minimum cost of any BST on $a_i, \dots, a_j$

Idea: The root of the BST can be any of $a_i, \dots, a_j$. We try each of them.

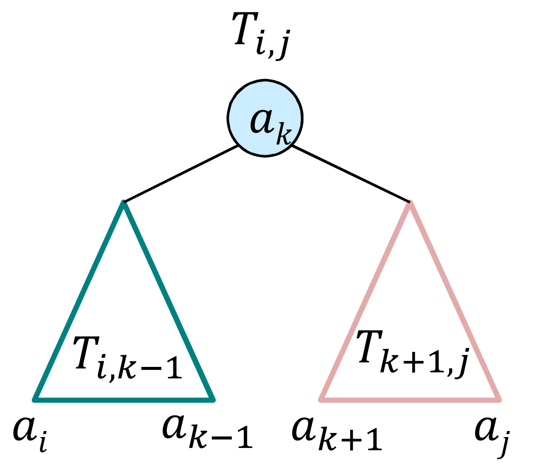
Recurrence:

Let $w[i,j] = f(a_i) + \cdots + f(a_j)$

Suppose we knew min-cost BST $T_{i,j}$ for [i,j] and that its root was a_k.

Its left subtree $T_{i,k-1}$ must be optimal for [i,k-1]
and its right subtree $T_{k+1,j}$ must be optimal for [k+1,j]

$$\Rightarrow \quad e[i,j] = B(T_{i,j})$$
$$= B(T_{i,k-1}) + B(T_{k+1,j}) + w[i,j]$$
$$= e[i,k-1] + e[k+1,j] + w[i,j]$$



To find $T_{i,j}$ we can try out every possible value of k and return the one which minimizes tree cost!

Dynamic Programming: The Recurrence

Def: $e[i,j]$ = the minimum cost of any BST on $a_i, \dots, a_j$

Idea: The root of the BST can be any of $a_i, \dots, a_j$. We try each of them.

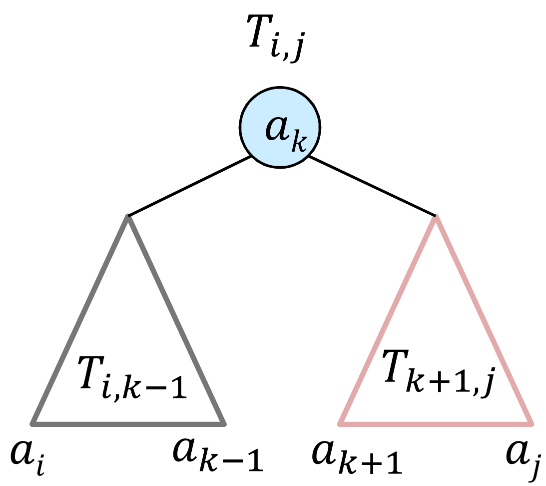
Recurrence:

Let $w[i,j] = f(a_i) + \cdots + f(a_j)$

$$e[i,j] = \min_{i \le k \le j} \{e[i,k-1] + e[k+1,j] + w[i,j]\}$$

$e[i,j] = 0$ for $i > j$.

$e[i,i] = f(a_i)$ for all i



Note: All $w[i,j]$'s can be pre-computed in $O(n^2)$ time.

The Algorithm

Idea: We will do the bottom-up computation by the increasing order of the problem size.

```
let e[1..n,1..n], w[1..n,1..n], root[1..n,1..n] be new arrays of all 0
for i = 1 to n                                   computation of w[i,j]
    w[i,i] ← f(a_i)
    for j = i+1 to n
        w[i,j] ← w[i,j−1] + f(a_j)
for l ← 1 to n                                   length of [i,j]
    for i ← 1 to n−l+1                           First node
        j ← i+l−1                                Last node
        e[i,j] ← ∞
        for k ← i to j                           Root k that
            t ← e[i,k−1] + e[k+1,j] + w[i,j]     minimizes
            if t < e[i,j] then                   e[i,k−1] + e[k+1,j] + w[i,j]
                e[i,j] ← t
                root[i,j] ← k
return Construct-BST(root,1,n)
```

Running time: $O(n^3)$

Space: $O(n^2)$

Construct the Optimal BST

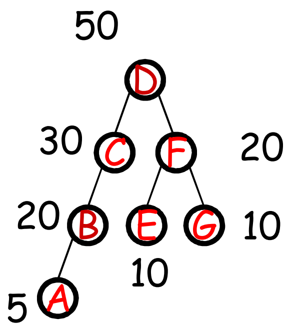
```
Construct-BST(root, i, j):
if i > j then return nil
create a node z
z.key ← a[root[i, j]]
z.left ← Construct-BST(root, i, root[i, j] − 1)
z.right ← Construct-BST(root, root[i, j] + 1, j)
return z
```

Running time of this part: $O(n)$

Worked example of the Optimal BST Problem

input

i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10



Optimal (min-cost) given input
Its cost is

5*4 + 20*3 + 30*2 + 50*1 + 10*3 + 20*2 + 10*3 = 290

Worked example of the Optimal BST Problem

input

i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

Precompute: $w[i,j] = f(a_i) + \cdots + f(a_j)$

Define: $e[i,j] =$ the minimum cost of any BST on $a_i, \ldots, a_j$

Recurrence: $e[i,j] = \min_{i \le k \le j} \{e[i,k-1] + e[k+1,j] + w[i,j]\}$

Initial Conditions: for $i > j$ $e[i,j] = 0$

and $e[i,i] = w[i,i] = f(a_i)$

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5						
2		20					
3			30				
4				50			
5					10		
6						20	
7							10

$$e[1,1] = f(a_1) = 5$$
$$e[2,2] = f(a_2) = 20$$
$$e[3,3] = f(a_3) = 30$$
$$e[4,4] = f(a_4) = 50$$
$$e[5,5] = f(a_5) = 10$$
$$e[6,6] = f(a_6) = 20$$
$$e[7\ 7] = f(a_7) = 10$$

$root[i,j]$

i \ j	1	2	3	4	5	6	7
1	1						
2		2					
3			3				
4				4			
5					5		
6						6	
7							7

- $root[1,1] = 1$
- $root[2,2] = 2$
- $root[3,3] = 3$
- $root[4,4] = 4$
- $root[5,5] = 5$
- $root[6,6] = 6$
- $root[7,7] = 7$

A B C D E F G

i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

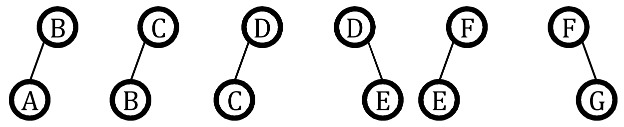
i \ j	1	2	3	4	5	6	7
1	5	30					
2		20	70				
3			30	110			
4				50	70		
5					10	40	
6						20	40
7							10

$$e[1,2] = \min_{1\le k\le 2}\{e[1,k-1] + e[k+1,2] + w[1,2]\} = \min\{45, 30\} = 30$$
$$e[2,3] = \min_{2\le k\le 3}\{e[2,k-1] + e[k+1,3] + w[2,3]\} = \min\{80, 70\} = 70$$
$$e[3,4] = \min_{2\le k\le 3}\{e[3,k-1] + e[k+1,4] + w[3,4]\} = \min\{130, 110\} = 110$$
$$e[4,5] = \min_{4\le k\le 5}\{e[4,k-1] + e[k+1,5] + w[4,5]\} = \min\{70, 110\} = 70$$
$$e[5,6] = \min_{5\le k\le 6}\{e[5,k-1] + e[k+1,6] + w[5,6]\} = \min\{50, 40\} = 40$$
$$e[6,7] = \min_{6\le k\le 7}\{e[6,k-1] + e[k+1,7] + w6[,7]\} = \min\{40, 50\} = 40$$

$root[i,j]$

i \ j	1	2	3	4	5	6	7
1	1	2					
2		2	3				
3			3	4			
4				4	4		
5					5	6	
6						6	6
7							7

- $root[1,2] = 2$
- $root[2,3] = 3$
- $root[3,4] = 4$
- $root[4,5] = 4$
- $root[5,6] = 6$
- $root[6,7] = 6$



i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5	30	85				
2		20	70	170			
3			30	110	130		
4				50	70	120	
5					10	40	60
6						20	40
7							10

$$e[1,3] = \min_{1\le k\le 3}\{e[1,k-1] + e[k+1,3] + w[1,3]\} = \min\{125, 90, 85\} = 85$$

$$e[2,4] = \min_{2\le k\le 4}\{210, 170, 170\} = 170$$

$$e[3,5] = \min_{3\le k\le 5}\{160, 130, 200\} = 130$$

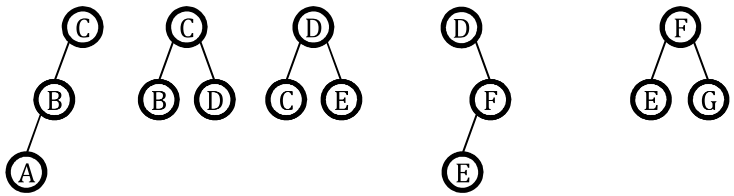
$$e[4,6] = \min_{4\le k\le 6}\{120, 150, 150\} = 120$$

$$e[5,7] = \min_{5\le k\le 7}\{80, 60, 80\} = 60$$

$root[i,j]$

i \ j	1	2	3	4	5	6	7
1	1	2	3				
2		2	3	3			
3			3	4	4		
4				4	4	4	
5					5	6	6
6						6	6
7							7

- $root[1,3] = 3$
- $root[2,4] = 3$
- $root[3,5] = 4$
- $root[4,6] = 4$
- $root[5,7] = 6$



i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5	30	85				
2		20	70	170			
3			30	110	130		
4				50	70	120	
5					10	40	60
6						20	40
7							10

$$e[1,4] = \min_{1\le k\le 4}\{e[1,k-1] + e[k+1,4] + w[1,4]\}$$

$$= \min_{1\le k\le 4}\left\{\begin{matrix} e[1,0]+e[2,4]+105, & e[1,1]+e[3,4]+105, \\ e[1,2]+e[4,4]+105, & e[1,3]+e[5,4]+105 \end{matrix}\right\}$$

$$= \min_{1\le k\le 4}\left\{\begin{matrix} 0+170+105, & 5+110+105, \\ 30+50+105, & 85+0+105 \end{matrix}\right\}$$

$$= \min\{275, 220, 185, 190\} = 185$$

$root[i,j]$

i \ j	1	2	3	4	5	6	7
1	1	2	3	3			
2		2	3	3			
3			3	4	4		
4				4	4	4	
5					5	6	6
6						6	6
7							7

i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5	30	85	185			
2		20	70	170	190		
3			30	110	130	180	
4				50	70	120	150
5					10	40	60
6						20	40
7							10

$$e[1,4] = \min_{1\le k\le 4}\{e[1,k-1]+e[k+1,4]+w[1,4]\} = \min\{275, 220, 185, 190\} = 185$$

$$e[2,5] = \min\ \{240, 200, 190, 280\} = 190$$

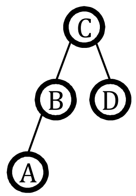
$$e[3,6] = \min\ \{230, 180, 240, 240\} = 180$$

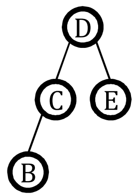
$$e[4,7] = \min\ \{150, 180, 170, 210\} = 150$$

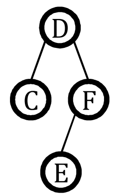
$root[i,j]$

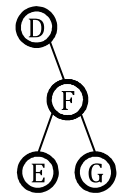
i \ j	1	2	3	4	5	6	7
1	1	2	3	3			
2		2	3	3	4		
3			3	4	4	4	
4				4	4	4	4
5					5	6	6
6						6	6
7							7

- $root[1,4] = 3$
- $root[2,5] = 4$
- $root[3,6] = 4$
- $root[4,7] = 4$









i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5	30	85	185	210		
2		20	70	170	190	240	
3			30	110	130	180	210
4				50	70	120	150
5					10	40	60
6						20	40
7							10

$$e[1,5] = \min_{1\le k\le 5}\{e[1,k-1] + e[k+1,5] + w[1,5]\} = \min\{305, 250, 215, 210, 300\} = 210$$

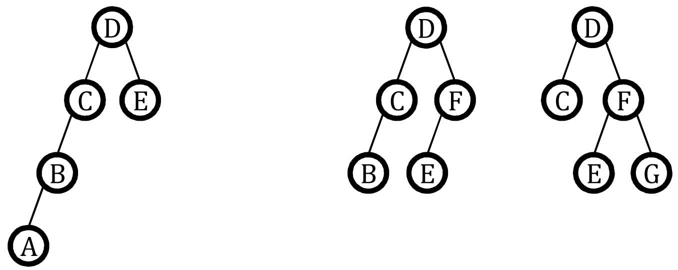
$$e[2,6] = \min\ \{310, 270, 240, 320, 320\} = 240$$

$$e[3,7] = \min\ \{270, 210, 270, 260, 300\} = 210$$

$root[i,j]$

i \ j	1	2	3	4	5	6	7
1	1	2	3	3	4		
2		2	3	3	4	4	
3			3	4	4	4	4
4				4	4	4	4
5					5	6	6
6						6	6
7							7

- $root[1,5] =$ 4
- $root[2,6] =$ 4
- $root[3,7] =$ 4



i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5	30	85	185	210	260	
2		20	70	170	190	240	270
3			30	110	130	180	210
4				50	70	120	150
5					10	40	60
6						20	40
7							10

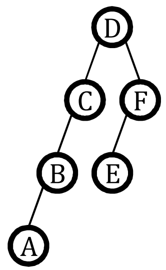
$$e[1,6] = \min_{1 \le k \le 6} \{e[1,k-1] + e[k+1,6] + w[1,6]\} = \min\{375, 320, 285, 260, 340, 345\} = 260$$

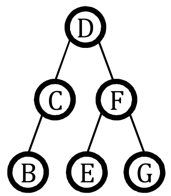
$$e[2,7] = \min\ \{350, 310, 270, 350, 340, 380\} = 270$$

$root[i,j]$

i \ j	1	2	3	4	5	6	7
1	1	2	3	3	4	4	
2		2	3	3	4	4	4
3			3	4	4	4	4
4				4	4	4	4
5					5	6	6
6						6	6
7							7

- $root[1,6] = 4$
- $root[2,7] = 4$





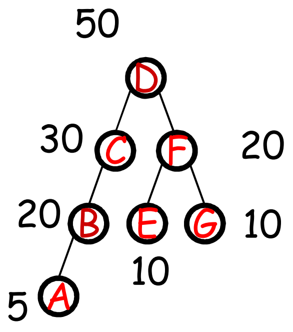
i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5	30	85	185	210	260	290
2		20	70	170	190	240	270
3			30	110	130	180	210
4				50	70	120	150
5					10	40	60
6						20	40
7							10

$$e[1,7] = \min_{1\le k\le 7}\{e[1,k-1] + e[k+1,7] + w[1,7]\} = \min\{415, 360, 325, 290, 370, 365, 405\} = 290$$

Optimal Tree: Cost = 290



$root[i,j]$

i \ j	1	2	3	4	5	6	7
1	1	2	3	3	4	4	4
2		2	3	3	4	4	4
3			3	4	4	4	4
4				4	4	4	4
5					5	6	6
6						6	6
7							7

- $root[1,7] = 4$

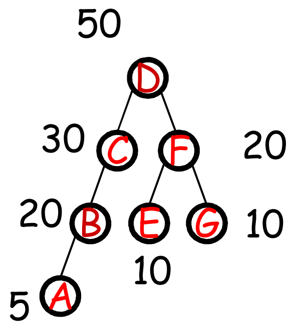
i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

$e[i,j]$

i \ j	1	2	3	4	5	6	7
1	5	30	85	185	210	260	290
2		20	70	170	190	240	270
3			30	110	130	180	210
4				50	70	120	150
5					10	40	60
6						20	40
7							10

$$e[1,7] = \min_{1 \le k \le 7}\{e[1,k-1] + e[k+1,7] + w[1,7]\} = \min\{415, 360, 325, 290, 370, 365, 405\} = 290$$

Optimal Tree: Cost = 290



$root[i,j]$

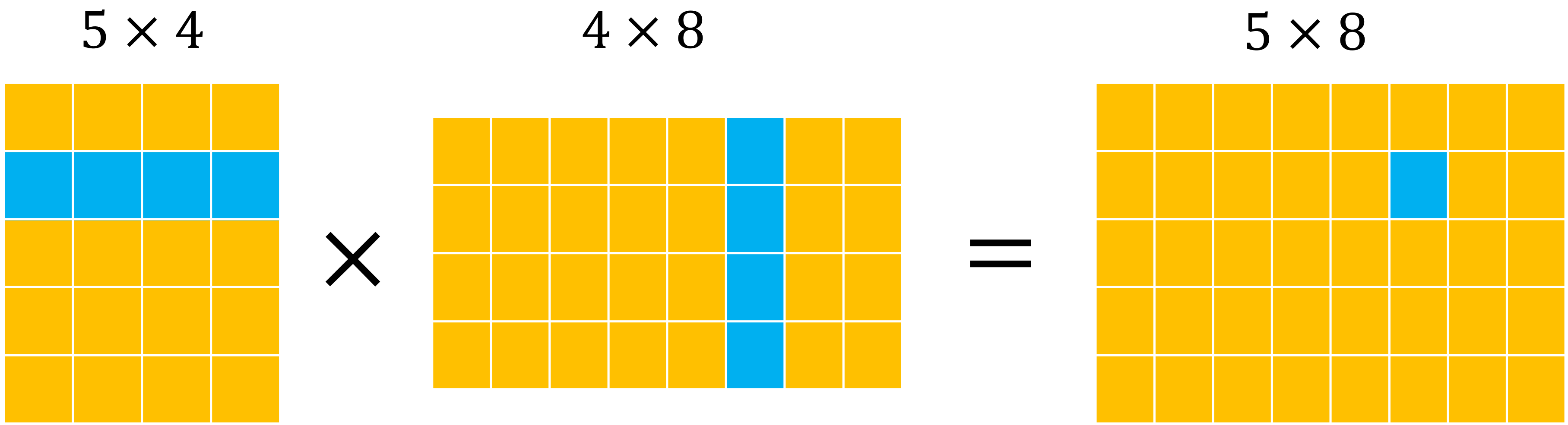
i \ j	1	2	3	4	5	6	7
1	1	2	3	3	4	4	4
2		2	3	3	4	4	4
3			3	4	4	4	4
4				4	4	4	4
5					5	6	6
6						6	6
7							7

- $root[1,7] = 4$ (D)
- $root[1,3] = 3$ (C)
- $root[5,7] = 6$ (F)
- $root[1,2] = 2$ (B)
- $root[1,1] = 1$ (A)
- $root[5,5] = 5$ (E)
- $root[7,7] = 7$ (G)

i	1	2	3	4	5	6	7
a_i	A	B	C	D	E	F	G
$f(a_i)$	5	20	30	50	10	20	10

Interval DP: Matrix-Chain Multiplication

The product of two matrices $A_{p\times q}$ and $B_{q\times r}$ (with dimensions $p \times q$ and $q \times r$) is a matrix $C_{p\times r}$.
Generating $C_{p\times r}$ requires pqr scalar multiplications.



Given matrices A, B with entries $a_{i,j}, b_{i,j}$, the entries in the product matrix $C = A \times B$ are

$$c_{i,j} = \sum_{k=1}^{q} a_{i,k} b_{k,j}$$

Diagrams on this and the next few pages modified from
http://ramos.elo.utfsm.cl/~lsb/elo320/aplicaciones/aplicaciones/CS460AlgorithmsandComplexity/lecture17

Matrix-Chain Multiplication - 3 Matrices

The product of two matrices $A_{p\times q}$ and $B_{q\times r}$ (with dimensions $p \times q$ and $q \times r$) is a matrix $C_{p\times r}$ with pr entries.

Calculating any one entry in $C_{p\times r}$ requires q scalar multiplications, so generating $C_{p\times r}$ requires pqr scalar multiplications (sm).

For three matrices (e.g., $A_{10\times 100}$, $B_{100\times 5}$ and $C_{5\times 50}$) there are 2 ways to parenthesize:

$(A(BC)) = A_{10\times 100}\cdot E_{100\times 50}$

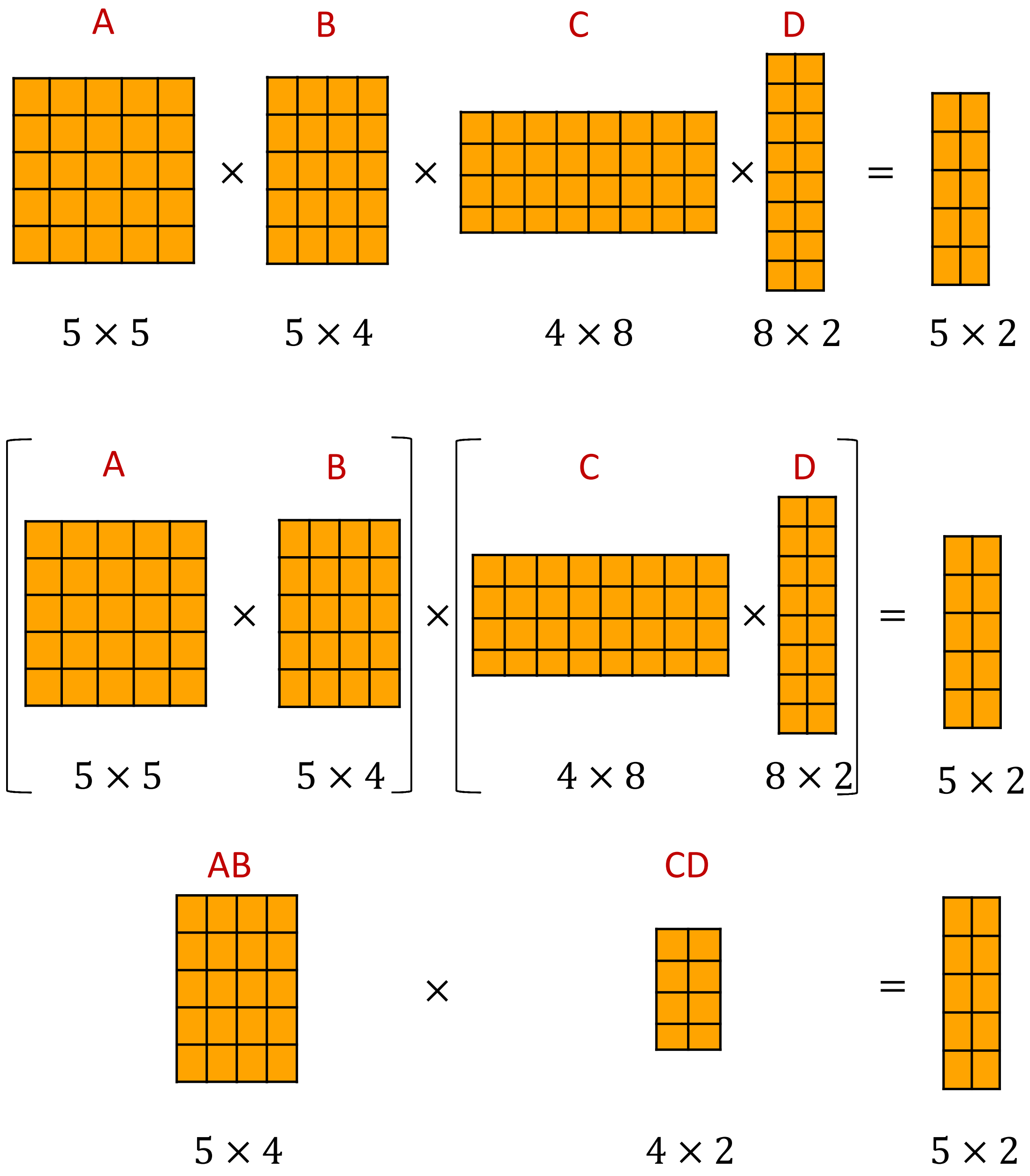
- $BC \Rightarrow 100\cdot 5\cdot 50 = 25{,}000$ sm
- $AE \Rightarrow 10\cdot 100\cdot 50 = 50{,}000$ sm
- Total $= 75{,}000$

$((AB)C) = D_{10\times 5}\cdot C_{5\times 50}$

- $AB \Rightarrow 10\cdot 100\cdot 5 = 5{,}000$ sm
- $DC \Rightarrow 10\cdot 5\cdot 50 = 2{,}500$ sm
- Total $= 7{,}500$

General problem: Given a sequence or chain $A_1, A_2, \ldots, A_n$, of n matrices, determine the optimal way to parenthesize (i.e., the solution with the minimum number of scalar multiplications).

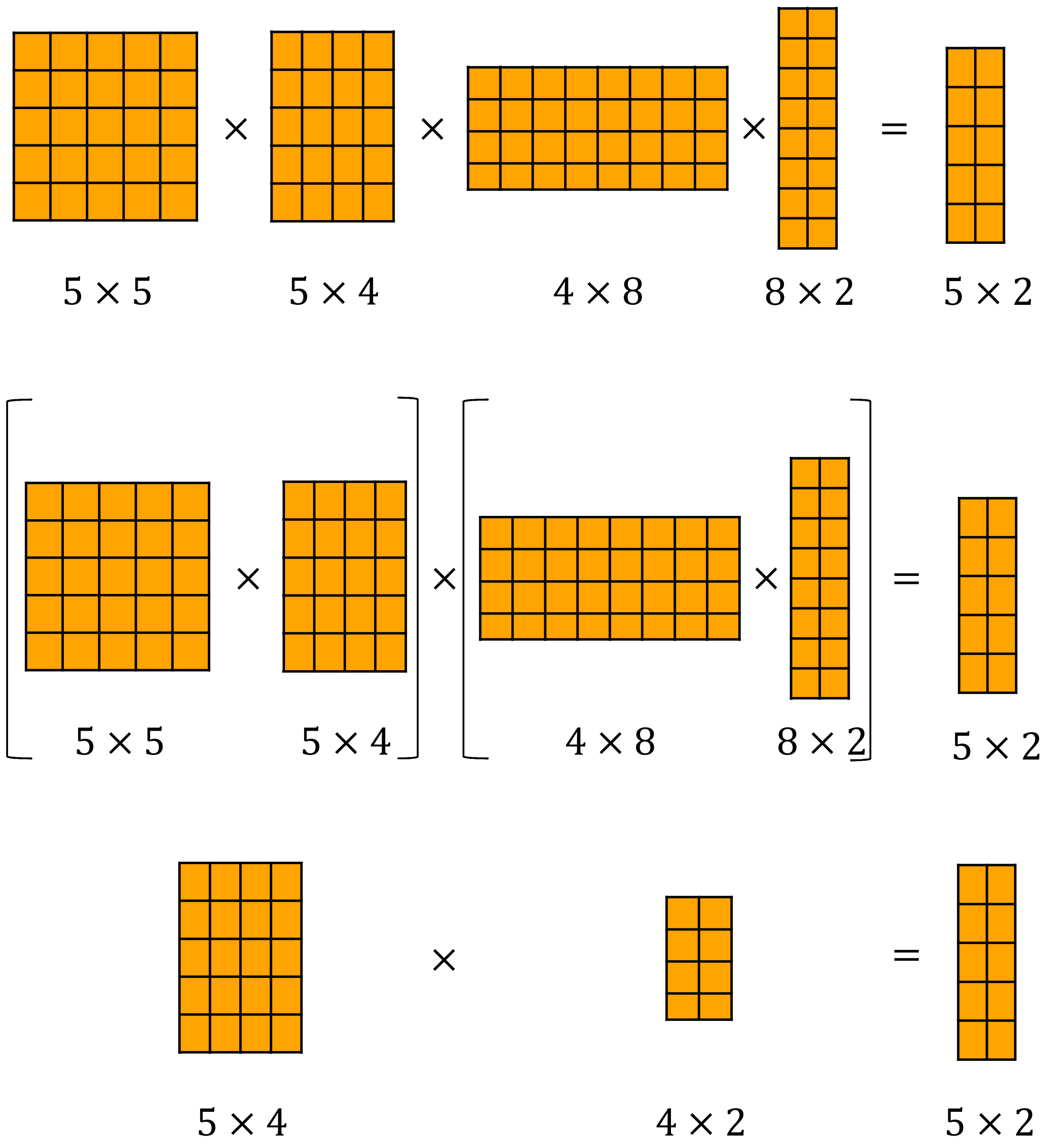
Matrix-Chain Multiplication - 5 Matrices



There are 5 different ways to multiply ABCD together

1. (A (B (CD)))
2. (A ((BC) D))
3. ((AB) (CD))
4. ((A (BC)) D)
5. (((AB) C) D)

Matrix-Chain Multiplication - 5 Matrices



There are 5 different ways to multiply ABCD together

1. (A (B (CD)))
2. (A ((BC) D))
3. ((AB) (CD))
4. ((A (BC)) D)
5. (((AB) C) D)

Costs are

1. 5(5)2 + 5(4)2 + 4(8)2 = 154
2. 5(5)2 + 5(4)8 + 5(8)2 = 290
3. 5(5)4 + 4(8)2 + 5(4)2 = 204
4. 5(5)8 + 5(4)8 + 5(8)2 = 440
5. 5(5)4 + 5(4)8 + 5(8)2 = 340

Recall: Multiplying
p×q and q×r matrices requires
p×q×r multiplications
And yields a p×r matrix

Problem Definition

- Input: Values $p_0\, p_1 \cdots p_{n-1}\, p_n$
- These represent sizes of n matrices $A_1\, A_2 \cdots A_n$
  Matrix A_i has dimensions $p_{i-1} \times p_i$
- $A_{i \dots j}$: matrix that is the product of $A_i\, A_{i+1} \cdots A_j$
  By construction $A_{i \dots j}$ has dimensions $p_{i-1} \times p_j$
- Goal: To find a minimum cost way of multiplying $A_1 A_2 \cdots A_n$ to get the final result $A_{1 \dots n}$.
  cost = # of total scalar multiplications performed

This is known as an optimal parenthesization of $A_1 A_2 \cdots A_n$ because the parentheses denote how to perform the multiplications e.g., ((AB) (CD)) means first calculate $X = AB$, then calculate $Y = CD$ and finally calculate XY.

Optimal Solution Structure

- Given: Values $p_0\, p_1 \cdots p_{n-1}\, p_n$ s.t. Matrix A_i has size $p_{i-1} \times p_i$
- $A_{i \cdots j}$: denotes matrix that results from the product $A_i A_{i+1} \cdots A_j$

- An (optimal) parenthesization of $A_1 A_2 \cdots A_n$ splits the product between A_k and A_{k+1} for some integer k where $1 \le k < n$.
  $A_{1 \cdots n} = (A_1 A_2 \cdots A_k) \cdot (A_{k+1} A_{k+2} \cdots A_n) = A_{1 \cdots k} \cdot A_{k+1 \cdots n}$

- In the optimal parenthesization,
  1st : compute matrices $A_{1 \cdots k}$ and $A_{k+1 \cdots n}$;
  2nd : multiply $A_{1 \cdots k}$ and $A_{k+1 \cdots n}$ together to get final matrix $A_{1 \cdots n}$

- **Observation**: If parenthesization of $A_1 A_2 \cdots A_n$ is optimal
  => parenthesizations of subchains $A_1 A_2 \cdots A_k$ and $A_{k+1} \cdots A_n$
  must also be optimal (why?)
  => The optimal solution to the problem contains within it the optimal solution to subproblems

Recurrence

- $m[i, j]$ =minimum number of scalar multiplications necessary to compute $A_{i \cdots j}$

- Suppose the optimal parenthesization of $A_{i \dots j}$ splits product between A_k and A_{k+1}, for some integer *k*, $\mathrm{i} \le k < \mathrm{j}$
  - $A_{i \dots j}$ = $(A_i A_{i+1} \cdots A_k) \cdot (A_{k+1} A_{k+2} \cdots A_j) = A_{i \dots k} \cdot A_{k+1 \cdots j}$
  - min cost of computing $A_{i \dots j}$ = min cost of computing $A_{i \dots k}$ + min cost of computing $A_{k+1 \dots j}$ + cost of multiplying $A_{i \dots k}$ and $A_{k+1 \dots j}$
  - Cost of multiplying $A_{i \dots k}$ and $A_{k+1 \dots j}$ is $p_{i-1} p_k p_j$

- But… optimal parenthesization occurs at *some* value of *k*. Check all possible values of *k* and select the best one.

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \} & \text{if } i < j \end{cases}$$

DP Algorithm

Input: *Array p*[0…*n*] containing matrix dimensions and *n*

Result: Minimum-cost table *m* and split table *s*
(s records values of k at which minima occurred)

MATRIX-CHAIN-ORDER(*p*[ ], *n*)
for *i* = 1 **to** *n*
    $m[i,i] = 0$
for $l = 2$ **to** *n*
    for $i = 1$ **to** $n - l + 1$
        $j = i + l - 1$
        $m[i,j] = \infty$
        for $k = i$ **to** $j - 1$
            $q = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
            if $q < m[i,j]$ **then**
                $m[i,j] = q$
                $s[i,j] = k$
return $m[\,]$ and $s[\,]$

Initial Conditions

Recurrence Relation

Location of Minimum

Time is $O(n^3)$, space is $O(n^2)$

Matrix Multiplication Worked Example

Input: $p_0\, p_1 \cdots p_{n-1}\, p_n$
These represent sizes of n matrices $A_1\ A_2 \cdots A_n$
Matrix A_i has dimensions $p_{i-1} \times p_i$

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

A_1: 5×4 A_2: 4×6 A_3: 6×2 A_4: 2×7 A_5: 7×4

$A_{i \cdots j}$: matrix product of $A_i\ A_{i+1} \cdots A_j$
$A_{i \cdots j}$ has dimensions $p_{i-1} \times p_j$

$m[i,j]$ = minimum number of scalar multiplications required to compute $A_{i \cdots j}$

Recurrence

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

$s[i,j]$ = records values of k at which minimum occurs

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

$\boldsymbol{m}[\boldsymbol{i},\boldsymbol{j}]$

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

$\boldsymbol{s}[\boldsymbol{i},\boldsymbol{j}]$

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

$l = 1$

$m[i,j]$

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

$s[i,j]$

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

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1}p_k p_j$$

$m[1,1] = 0$ A_1

$m[2,2] = 0$ A_2

$m[3,3] = 0$ A_3

$m[4,4] = 0$ A_4

$m[5,5] = 0$ A_5

$$l = 2$$

$m[i,j]$

i \ j	1	2	3	4	5
1	0	120			
2		0	48		
3			0	84	
4				0	56
5					0

$s[i,j]$

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

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

$m[1,2] = m[1,1] + m[2,2] + 5 * 4 * 6 = 120$ $\quad A_1A_2$

- $m[2,3] = m[2,2] + m[3,3] + 4 * 6 * 2 = 48$ $\quad A_2A_3$
- $m[3,4] = m[3,3] + m[4,4] + 6 * 2 * 7 = 84$ $\quad A_3A_4$
- $m[4,5] = m[4,4] + m[5,5] + 2 * 7 * 4 = 56$ $\quad A_4A_5$

$l = 3$

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1}p_k p_j$$

$m[i,j]$

i \ j	1	2	3	4	5
1	0	120	88		
2		0	48	104	
3			0	84	104
4				0	56
5					0

$$m[1,3] = \min \begin{Bmatrix} m[1,1] + m[2,3] + 5*4*2, \\ m[1,2] + m[3,3] + 5*6*2 \end{Bmatrix} = 88 \qquad A_1 A_{2..3}$$

$s[i,j]$

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

$$m[2,4] = \min \begin{Bmatrix} m[2,2] + m[3,4] + 4*6*7, \\ m[2,3] + m[4,4] + 4*2*7 \end{Bmatrix} = 104 \qquad A_{2..3} A_4$$

$$m[3,5] = \min \begin{Bmatrix} m[3,3] + m[4,5] + 6*2*4, \\ m[3,4] + m[5,5] + 6*7*4 \end{Bmatrix} = 104 \qquad A_3 A_{4..5}$$

$$l = 4$$

$\boldsymbol{m[i,j]}$

i \ j	1	2	3	4	5
1	0	120	88	158	
2		0	48	104	136
3			0	84	104
4				0	56
5					0

$\boldsymbol{s[i,j]}$

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

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

$$m[1,4] = \min \begin{Bmatrix} m[1,1] + m[2,4] + 5 * 4 * 7 \\ m[1,2] + m[3,4] + 5 * 6 * 7 \\ \boldsymbol{m[1,3] + m[4,4] + 5 * 2 * 7} \end{Bmatrix} = 158$$

$\boldsymbol{A_{1..3}A_4}$

$$m[2,5] = \min \begin{Bmatrix} m[2,2] + m[3,5] + 4 * 6 * 4 \\ \boldsymbol{m[2,3] + m[4,5] + 4 * 2 * 4} \\ m[2,4] + m[5,5] + 4 * 7 * 4 \end{Bmatrix} = 136$$

$\boldsymbol{A_{2..3}A_{4..5}}$

$l = 5$

$m[i,j]$

i \ j	1	2	3	4	5
1	0	120	88	158	184
2		0	48	104	136
3			0	84	104
4				0	56
5					0

$s[i,j]$

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

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

$$m[1,5] = \max \begin{Bmatrix} m[1,4] + m[5,5] + 5 * 7 * 4, \\ \boldsymbol{m[1,3] + m[4,5] + 5 * 2 * 4}, \\ m[1,2] + m[3,5] + 5 * 6 * 4, \\ m[1,1] + m[2,5] + 5 * 4 * 4 \end{Bmatrix} = 184$$

$\boldsymbol{A_{1..3} A_{4..5}}$

$l = 5$

$m[i,j]$

i \ j	1	2	3	4	5
1	0	120	88	158	184
2		0	48	104	136
3			0	84	104
4				0	56
5					0

$s[i,j]$

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

i	0	1	2	3	4	5
p_i	5	4	6	2	7	4

$$m[i,i] = 0$$
$$m[i,j] = \min_{i \le k < j} [i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

Optimal solution is

$$(A_{1..s[1,5]} A_{s[1,5]+1..5}) = (A_{1..3} A_{4..5})$$
$$= (A_{1..s[1,3]} A_{s[1,3]+1..5}) A_{4..5} \quad = ((A_{1..1} A_{2..3}) A_{4..5})$$
$$= ((A_1 (A_{2..s[2,3]} A_{s[2,3]+1..3})) A_{4..5})$$
$$= ((A_1 (A_{2..2} A_{3..3})) A_{4..5})$$
$$= ((A_1 (A_2 A_3)) A_{4..5})$$
$$= ((A_1 (A_2 A_3)) (A_{4..s[4,5]} A_{s[4,5]+1,5}))$$
$$= ((A_1 (A_2 A_3)) (A_4 A_5))$$