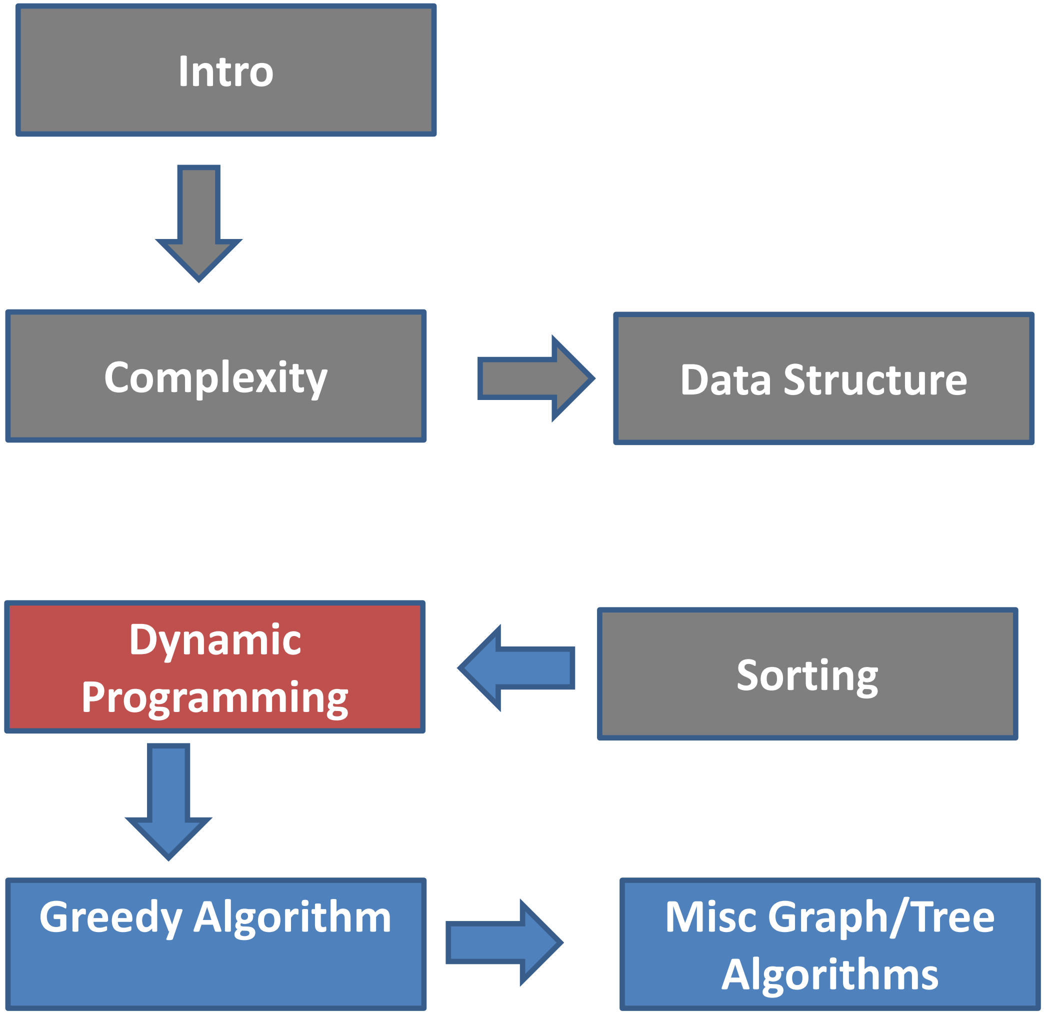
An Introduction to Algorithms

By

Hossein Rahmani

h_rahmani@iust.ac.ir
http://webpages.iust.ac.ir/h_rahmani/



Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms

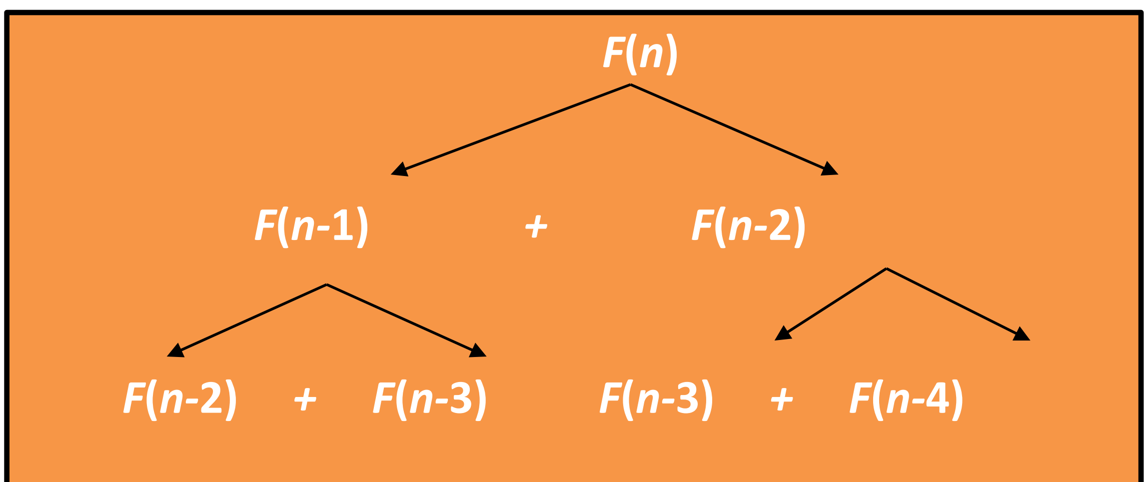
Those who cannot remember the past are condemned to repeat it.

-Dynamic Programming

Fibonacci Numbers

- Computing the n^{th} Fibonacci number recursively:
  - F(n) = F(n-1) + F(n-2)
  - F(0) = 0
  - F(1) = 1
  - Top-down approach

```
int Fib(int n)
{
    if (n <= 1)
        return 1;
    else
        return Fib(n - 1) + Fib(n - 2);
}
```



Fibonacci Numbers

- What is the Recurrence relationship?
  - $T(n) = T(n-1) + T(n-2) + 1$
- What is the solution to this?
  - Clearly it is $O(2^n)$
  - You should notice that $T(n)$ grows very similarly to $F(n)$, so in fact $T(n) = \Theta(F(n))$.
- Obviously not very good, but we know that there is a better way to solve it!

Fibonacci Numbers

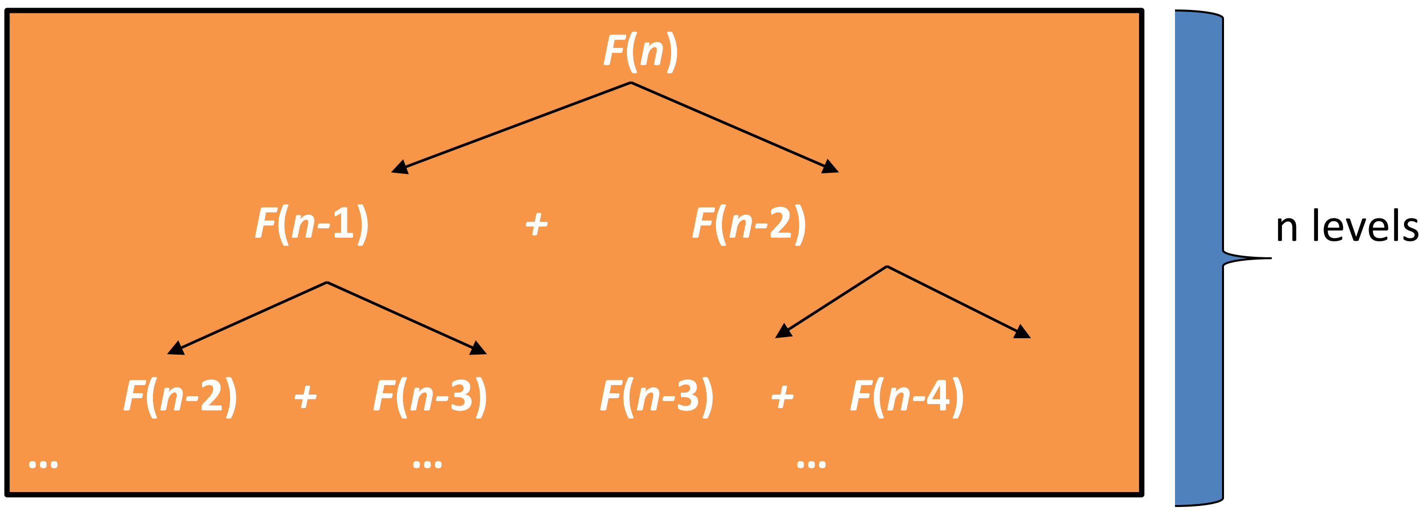
- Computing the $n^{\underline{th}}$ Fibonacci number using a bottom-up approach:
  - F(0) = 0
  - F(1) = 1
  - F(2) = 1+0 = 1
  - …
  - F(n-2) =
  - F(n-1) =
  - F(n) = F(n-1) + F(n-2)

0	**1**	**1**	**. . .**	***F*(*n*-2)**	***F*(*n*-1)**	***F*(*n*)**

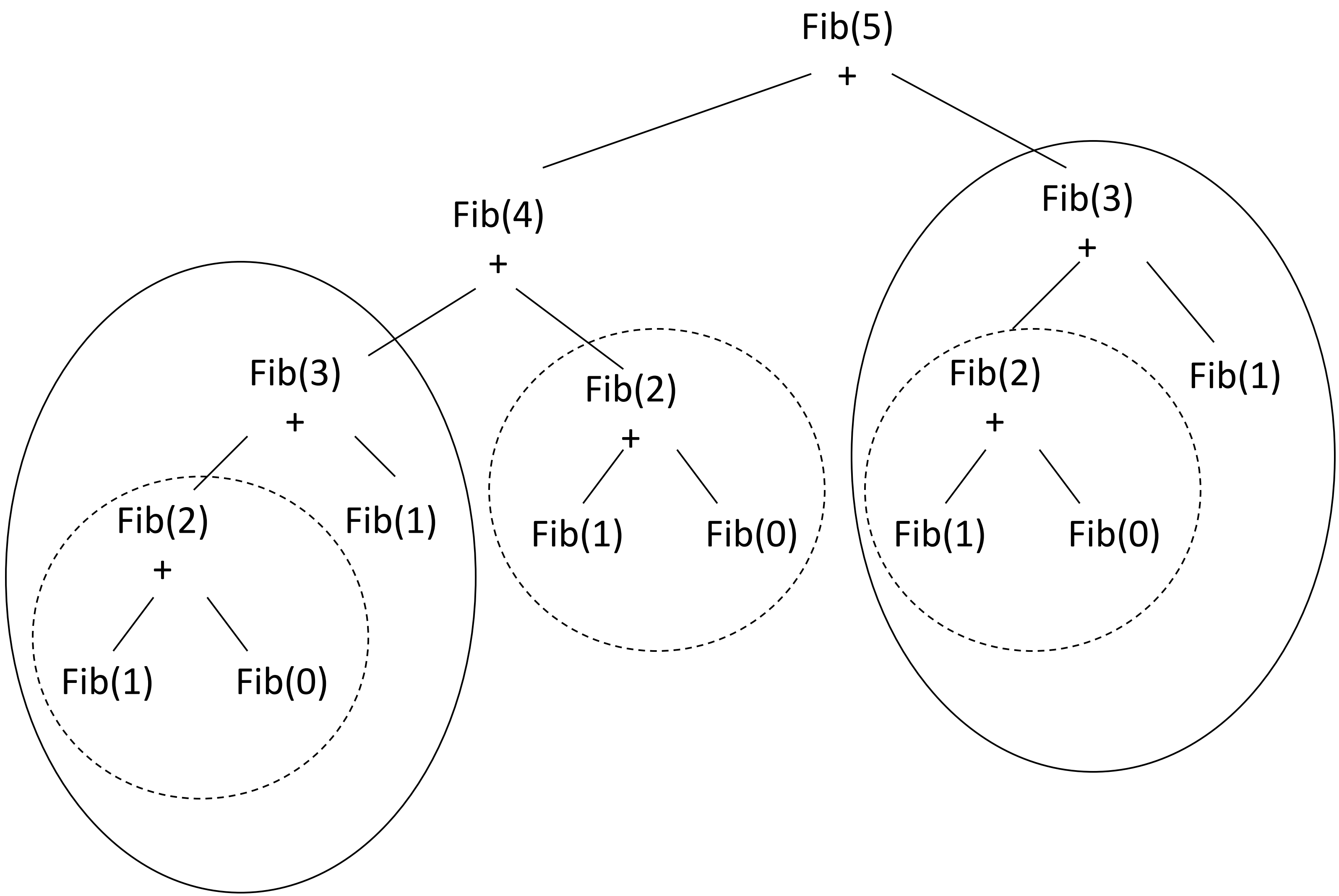
- Efficiency:
  - Time – O(n)
  - Space – O(n)

Fibonacci Numbers

- The bottom-up approach is only $\Theta(n)$.
- Why is the top-down so inefficient?
  - Re-computes many sub-problems.
    - How many times is F(n-5) computed?



Fibonacci Numbers



Dynamic Programming

- Dynamic Programming is an algorithm design technique for *optimization problems:* often minimizing or maximizing.
- Like divide and conquer, DP solves problems by combining solutions to sub-problems.
- Unlike divide and conquer, sub-problems are not independent.
  - Sub-problems may share sub-problems,

Dynamic Programming

- The term Dynamic Programming comes from Control Theory, not computer science. Programming refers to the use of tables (arrays) to construct a solution.
- In dynamic programming we usually reduce time by increasing the amount of space
- We solve the problem by solving sub-problems of increasing size and saving each optimal solution in a table (usually).
- The table is then used for finding the optimal solution to larger problems.
- Time is saved since each sub-problem is solved only once.

Dynamic Programming

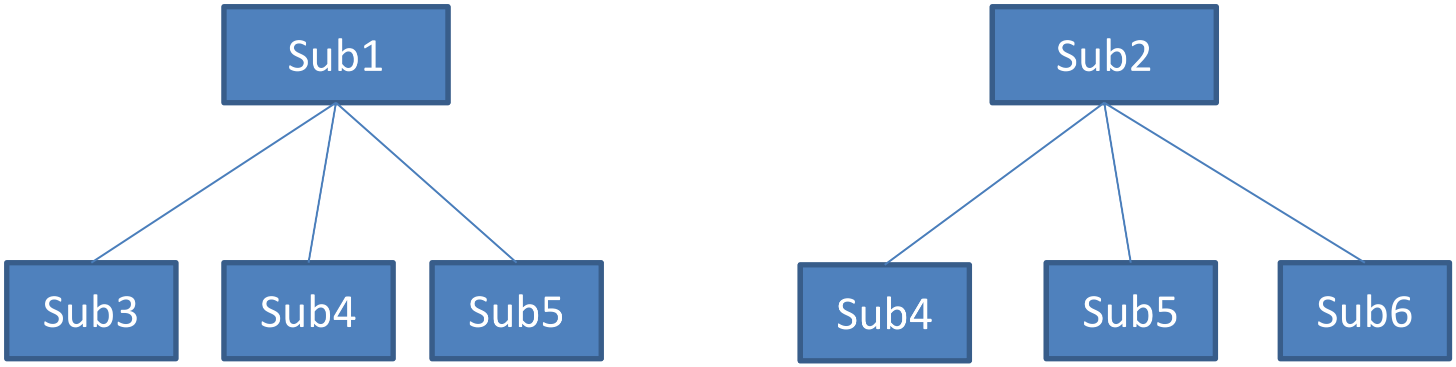
- How dynamic programming (DP) works?
  - Approach to solve problems
  - Store partial solutions of the smaller problems
  - Usually they are solved bottom-up
- Steps to designing a DP algorithm:
  1. Characterize optimal substructure
  2. Recursively define the value of an optimal solution
  3. Compute the value bottom up
  4. (if needed) Construct an optimal solution

Elements of DP

- DP has the following characteristics
  - Simple subproblems
    - We break the original problem to smaller sub-problems that have the same structure
  - Optimal substructure of the problems
    - The optimal solution to the problem contains within optimal solutions to its sub-problems
  - Overlapping sub-problems
    - There exist some places where we solve the same sub-problem more than once

What is dynamic programming?

- Sub-problems overlap
  - Sub-problems share sub-problems



What is dynamic programming?

- Dynamic programming algorithms
  - Solve each subproblem only **once**
  - Record sub problem solutions in a table

Difference between DP and Divide-and-Conquer

- Using Divide-and-Conquer to solve problems (that can be solved using DP) is inefficient
  - Because the same common sub-problems have to be solved many times
- DP will solve each of them once and their answers are stored in a table for future use
  - Technique known as "memoization"
    - In computing, memoization is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again

Dynamic Programming

- The best way to get a feel for this is through some more examples.
  - Matrix Chaining optimization
  - Longest Common Subsequence
  - 0-1 Knapsack Problem
  - Transitive Closure of a direct graph

Matrix chain multiplication

- **What is the number of real number multiplication needed for multiplying 2 matrices?**
  - $A_{pq} \times B_{qr} = C_{pr}$
  - pqr
- **For a matrix chain, do you think the order to multiplication matters?**

  $(A_1(A_2A_3))$

  $((A_1A_2)A_3)$

  - The order does not affect the result
  - But affects the cost (the number of real number multiplications)

Matrix chain multiplication

- $M_{6\times 2}\ M_{2\times 5}\ M_{5\times 20}$
- $(M_{6\times 2}M_{2\times 5})M_{5\times 20}$
  - Cost = $6\times 2\times 5+6\times 5\times 20$

    $=60+600=660$
- $M_{6\times 2}(M_{2\times 5}M_{5\times 20})$
  - Cost = $6\times 2\times 20+2\times 5\times 20$

    $=240+200=440$
- **With different <u>parenthesizations</u>, <u>costs</u> are different**

Matrix chain multiplication

- **Matrix chain multiplication problem**: given a matrix chain, find out a parenthesization with the lowest cost
- How to solve it by brute force?

$$A_1 \quad A_2 \quad A_3 \quad \ldots \quad A_n$$

Matrix chain multiplication

- How to solve it buy brute force?

$(A_1 (A_2 (A_3 A_4)))$

How to divide and conquer?

$(A_1 ((A_2 A_3) A_4))$

Last multiplication is between A1 and A2

$((A_1 A_2) (A_3 A_4))$

Last multiplication is between A2 and A3

$((A_1 (A_2 A_3)) A_4)$

Last multiplication is between A3 and A4

$(((A_1 A_2) A_3) A_4)$

Matrix chain multiplication

- How to solve it buy divide-and-conquer?

$\boxed{A_1}\boxed{A_2}\boxed{A_3}\boxed{A_4}$

optimal cost of $\boxed{A_1}$ + optimal cost of $\boxed{A_2}\boxed{A_3}\boxed{A_4}$ + cost of $\boxed{A_1}\boxed{A_{24}}$

optimal cost of $\boxed{A_1}\boxed{A_2}$ + optimal cost of $\boxed{A_3}\boxed{A_4}$ + cost of $\boxed{A_{12}}\boxed{A_{34}}$

optimal cost of $\boxed{A_1}\boxed{A_2}\boxed{A_3}$ + optimal cost of $\boxed{A_4}$ + cost of $\boxed{A_{13}}\boxed{A_4}$

Choose the smallest one from them Weakness?

Many subproblems are overlapped

Matrix chain multiplication

A_1 A_2 A_3 … A_n

There are n-1 ways to divide it into 2 smaller matrix chains, if divide it after matrix k, for each of them we need to calculate:

optimal cost of A_1 … A_k + optimal cost of A_{k+1} … A_n + cost of A_{1k} $A_{k+1,n}$

Choose the smallest one from them

Many subproblems are overlapped

Matrix chain multiplication

- How do it in a dynamic programming way?
  - Top-down, <u>record the solutions to subproblem</u>
  - Or, bottom-up, start from smallest problems (**the typical manner**)
    - Solve all the possible subproblems

Let's try it!

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	A_1	A_1A_2	$A_1 \ldots A_3$	$A_1 \ldots A_4$	$A_1 \ldots A_5$	$A_1 \ldots A_6$
2		A_2	A_2A_3	$A_2 \ldots A_4$	$A_2 \ldots A_5$	$A_2 \ldots A_6$
3			A_3	A_3A_4	$A_3 \ldots A_5$	$A_3 \ldots A_6$
4				A_4	A_4A_5	$A_4 \ldots A_6$
5					A_5	A_5A_6
6						A_6

Table $Cell_{ij}$ records the minimal cost of $A_i \ldots A_j$

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0 A_1	15750 A_1A_2	$A_1 \dots A_3$	$A_1 \dots A_4$	$A_1 \dots A_5$	$A_1 \dots A_6$
2		0 A_2	2625 A_2A_3	$A_2 \dots A_4$	$A_2 \dots A_5$	$A_2 \dots A_6$
3			0 A_3	750 A_3A_4	$A_3 \dots A_5$	$A_3 \dots A_6$
4				0 A_4	1000 A_4A_5	$A_4 \dots A_6$
5					0 A_5	5000 A_5A_6
6						0 A_6

30*35*15=15750

35*15*5=2625

15*5*10=750

5*10*20=1000

10*20*25=5000

Table $Cell_{ij}$ records the minimal cost of A_i … A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875, 1-2	$A_1 \dots A_4$	$A_1 \dots A_5$	$A_1 \dots A_6$
2		0	2625	$A_2 \dots A_4$	$A_2 \dots A_5$	$A_2 \dots A_6$
3			0	750	$A_3 \dots A_5$	$A_3 \dots A_6$
4				0	1000	$A_4 \dots A_6$
5					0	5000
6						0

A_1 A_2 A_3

0+2625+30*35*5=7875

A_1 A_2 A_3

15750+0+30*15*5=30825

Table $Cell_{ij}$ records the minimal cost of A_i ... A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	$A_1 \dots A_4$	$A_1 \dots A_5$	$A_1 \dots A_6$
2		0	2625	4375, 3-4	$A_2 \dots A_5$	$A_2 \dots A_6$
3			0	750	$A_3 \dots A_5$	$A_3 \dots A_6$
4				0	1000	$A_4 \dots A_6$
5					0	5000
6						0

A_2 $\quad$ A_3 A_4

0+750+35*15*10=6000

A_2 A_3 $\quad$ A_4

2625+0+35*5*10=4375

Table $Cell_{ij}$ records the minimal cost of A_i … A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	$A_1 \ldots A_4$	$A_1 \ldots A_5$	$A_1 \ldots A_6$
2		0	2625	4375, 3-4	$A_2 \ldots A_5$	$A_2 \ldots A_6$
3			0	750	$A_3 \ldots A_5$ 2500, 3-4	$A_3 \ldots A_6$
4				0	1000	$A_4 \ldots A_6$
5					0	5000
6						0

A_3 A_4 A_5

0+1000+15*5*20=2500

A_3 A_4 A_5

750+0+15*10*20=3750

Table $Cell_{ij}$ records the minimal cost of A_i … A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	$A_1 \ldots A_4$	$A_1 \ldots A_5$	$A_1 \ldots A_6$
2		0	2625	4375, 3-4	$A_2 \ldots A_5$	$A_2 \ldots A_6$
3			0	750	2500, 3-4	$A_3 \ldots A_6$
4				0	1000	3500, 5-6
5					0	5000
6						0

A_4 A_5 A_6

0+5000+5*10*25=6250

A_4 A_5 A_6

1000+0+5*20*25=3500

Table $Cell_{ij}$ records the minimal cost of A_i ... A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	9375, 3-4	$A_1 \dots A_5$	$A_1 \dots A_6$
2		0	2625	4375, 3-4	$A_2 \dots A_5$	$A_2 \dots A_6$
3			0	750	2500, 3-4	$A_3 \dots A_6$
4				0	1000	3500, 5-6
5					0	5000
6						0

A_1 A_2 A_3 A_4

0+4375+30*35*10=14875

A_1 A_2 A_3 A_4

15750+750+30*15*10=21000

A_1 A_2 A_3 A_4

7875+0+30*5*10=9375

Table $Cell_{ij}$ records the minimal cost of A_i ... A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	9375, 3-4	$A_1 \ldots A_5$	$A_1 \ldots A_6$
2		0	2625	4375, 3-4	$A_2 \ldots A_5$ 7125, 3-4	$A_2 \ldots A_6$
3			0	750	2500, 3-4	$A_3 \ldots A_6$
4				0	1000	3500, 5-6
5					0	5000
6						0

A_2 | A_3 A_4 A_5

0+2500+35*15*20=13000

A_2 A_3 | A_4 A_5

2625+1000+35*5*20=7125

A_2 A_3 A_4 | A_5

4375+0+35*10*20=11375

Table $Cell_{ij}$ records the minimal cost of A_i … A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	9375, 3-4	$A_1 \ldots A_5$	$A_1 \ldots A_6$
2		0	2625	4375, 3-4	7125, 3-4	$A_2 \ldots A_6$
3			0	750	2500, 3-4	5375, 3-4
4				0	1000	3500, 5-6
5					0	5000
6						0

A_3 $\quad$ $A_4\ A_5\ A_6$

0+3500+15*5*25=5375

$A_3\ A_4$ $\quad$ $A_5\ A_6$

750+5000+15*10*25=9500

$A_3\ A_4\ A_5$ $\quad$ A_6

2500+0+15*20*25=10000

Table $Cell_{ij}$ records the minimal cost of $A_i \ \ldots \ A_j$

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	9375, 3-4	11875, 3-4	$A_1 \dots A_6$
2		0	2625	4375, 3-4	7125, 3-4	$A_2 \dots A_6$
3			0	750	2500, 3-4	5375, 3-4
4				0	1000	3500, 5-6
5					0	5000
6						0

A_1 A_2 A_3 A_4 A_5

0+7125+30*35*20=28125

A_1 A_2 A_3 A_4 A_5

15750+2500+30*15*20=27250

A_1 A_2 A_3 A_4 A_5

7875+1000+30*5*20=11875

A_1 A_2 A_3 A_4 A_5

9375+0+30*20*25=24375

Table $Cell_{ij}$ records the minimal cost of A_i … A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	9375, 3-4	11875, 3-4	$A_1 \dots A_6$
2		0	2625	4375, 3-4	7125, 3-4	18125, 3-4
3			0	750	2500, 3-4	5375, 3-4
4				0	1000	3500, 5-6
5					0	5000
6						0

A_2 A_3 A_4 A_5 A_6

0+5375+35*15*25=18500

A_2 A_3 A_4 A_5 A_6

2625+3500+35*5*25=10500

A_2 A_3 A_4 A_5 A_6

4375+5000+35*10*25=18125

A_2 A_3 A_4 A_5 A_6

7125+0+35*20*25=24625

Table $Cell_{ij}$ records the minimal cost of A_i … A_j

Matrix chain multiplication

A_1	A_2	A_3	A_4	A_5	A_6
30*35	35*15	15*5	5*10	10*20	20*25

	1	2	3	4	5	6
1	0	15750	7875 ,1-2	9375, 3-4	11875, 3-4	15125, 3-4
2		0	2625	4375, 3-4	7125, 3-4	18125, 3-4
3			0	750	2500, 3-4	5375, 3-4
4				0	1000	3500, 5-6
5					0	5000
6						0

A_1 | A_2 A_3 A_4 A_5 A_6

0+18125+30*35*25=44375

A_1 A_2 | A_3 A_4 A_5 A_6

15750+5375+30*15*25=32375

A_1 A_2 A_3 | A_4 A_5 A_6

7875+3500+30*5*25=15125

A_1 A_2 A_3 A_4 | A_5 A_6

9375+5000+30*10*25=21875

A_1 A_2 A_3 A_4 A_5 | A_6

11875+0+30*20*25=26875

Table $Cell_{ij}$ records the minimal cost of A_i … A_j

Matrix chain multiplication

	1	2	3	4	5	6
1	0	15750	7875, 1-2	9375, 3-4	11875, 3-4	15125, 3-4
2		0	2625	4375, 3-4	7125, 3-4	18125, 3-4
3			0	750	2500, 3-4	5375, 3-4
4				0	1000	3500, 5-6
5					0	5000
6						0

$(A_1 (A_2 \; A_3)) ((A_4 \; A_5) A_6)$

Time complexity?

Number of subproblems = $\frac{n(n-1)}{2} = O(n^2)$

When all sub solutions are known, time needed for a solution=O(n)

So T(n) = $O(n^3)$

Quiz 1
Matrix Chain

A_1	A_2	A_3	A_4
3*2	2*5	5*10	10*6

Quiz 2

- Four matrices M1, M2, M3 and M4 of dimensions pxq, qxr, rxs and sxt respectively can be multiplied is several ways with different number of total scalar multiplications. For example, when multiplied as ((M1 X M2) X (M3 X M4)), the total number of multiplications is pqr + rst + prt. If p = 10, q = 100, r = 20, s = 5 and t = 80, then the number of scalar multiplications needed is:

Solution to Quiz 1

A_1	A_2	A_3	A_4
3*2	2*5	5*10	10*6

	1	2	3	4
1	0	30	160, 1-2	256, 1-2
2		0	100	220, 3-4
3			0	300
4				0