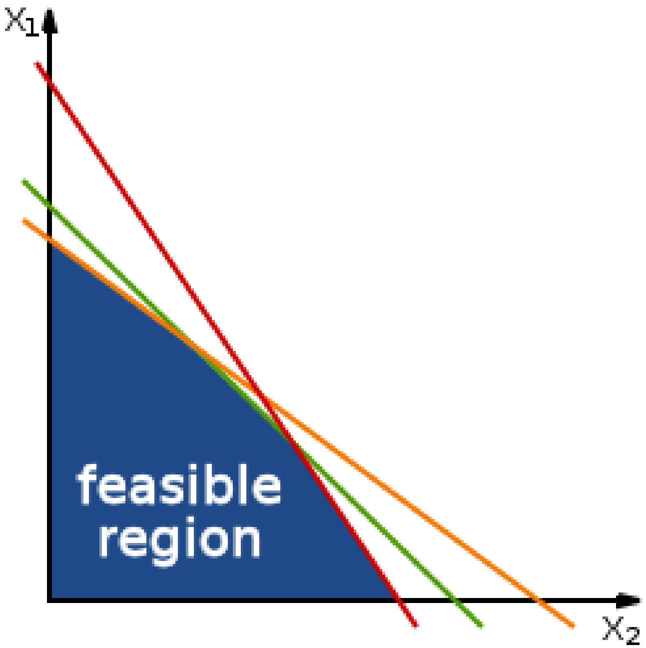
CS141: Intermediate Data Structures and Algorithms

Dynamic Programming

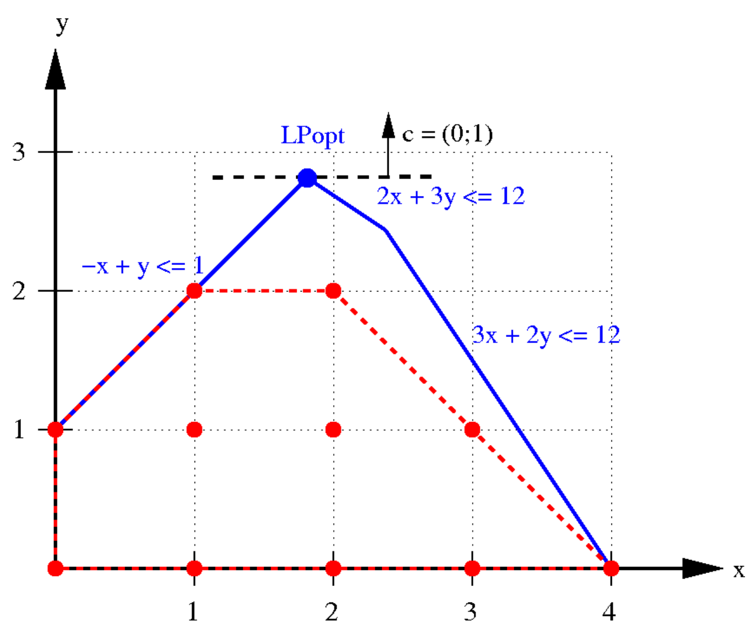
Amr Magdy

Programming?

- In this context, programming is a tabular method
  - Storing previously calculated results in a table, and look it up later

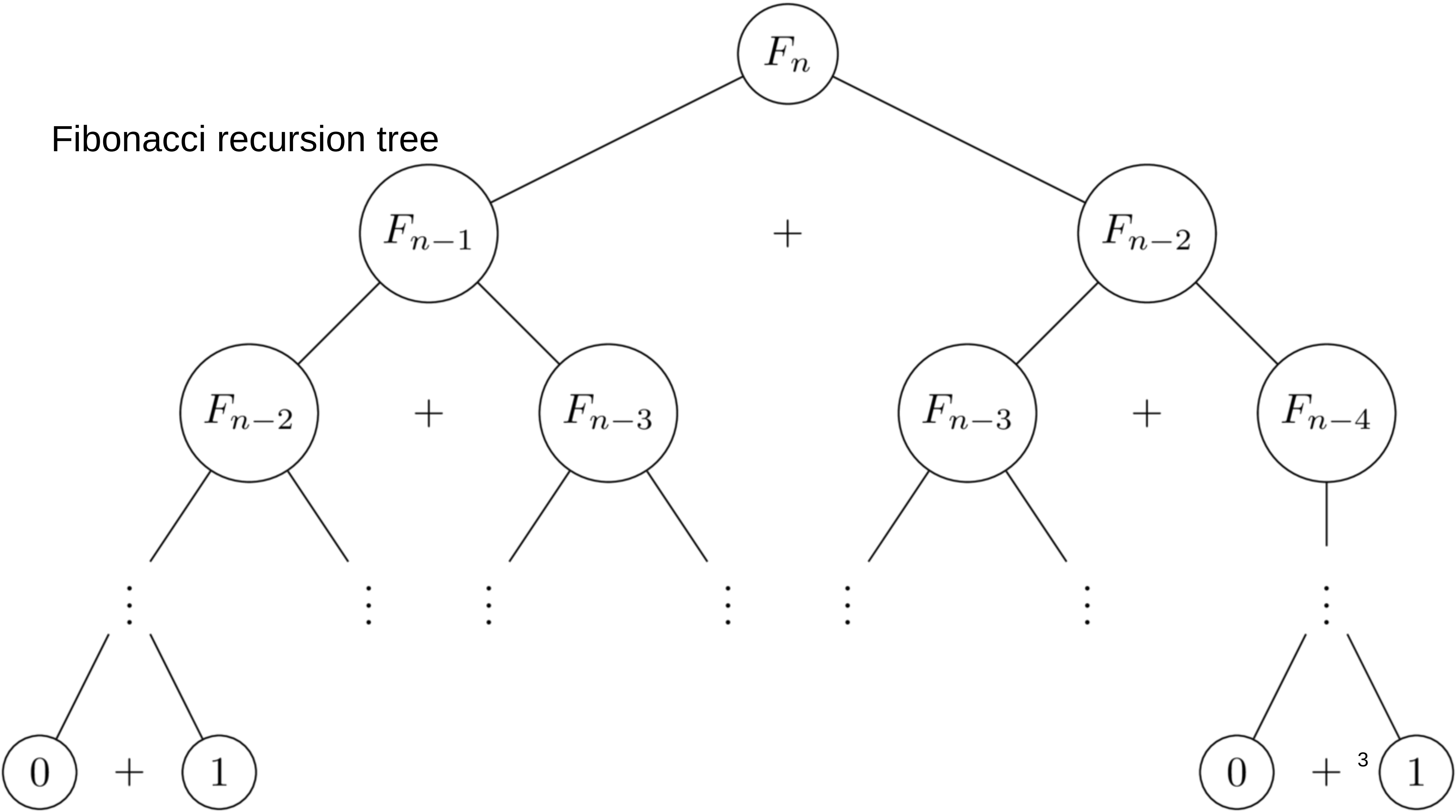


- Other examples:
  - Linear programing
  - Integer programming



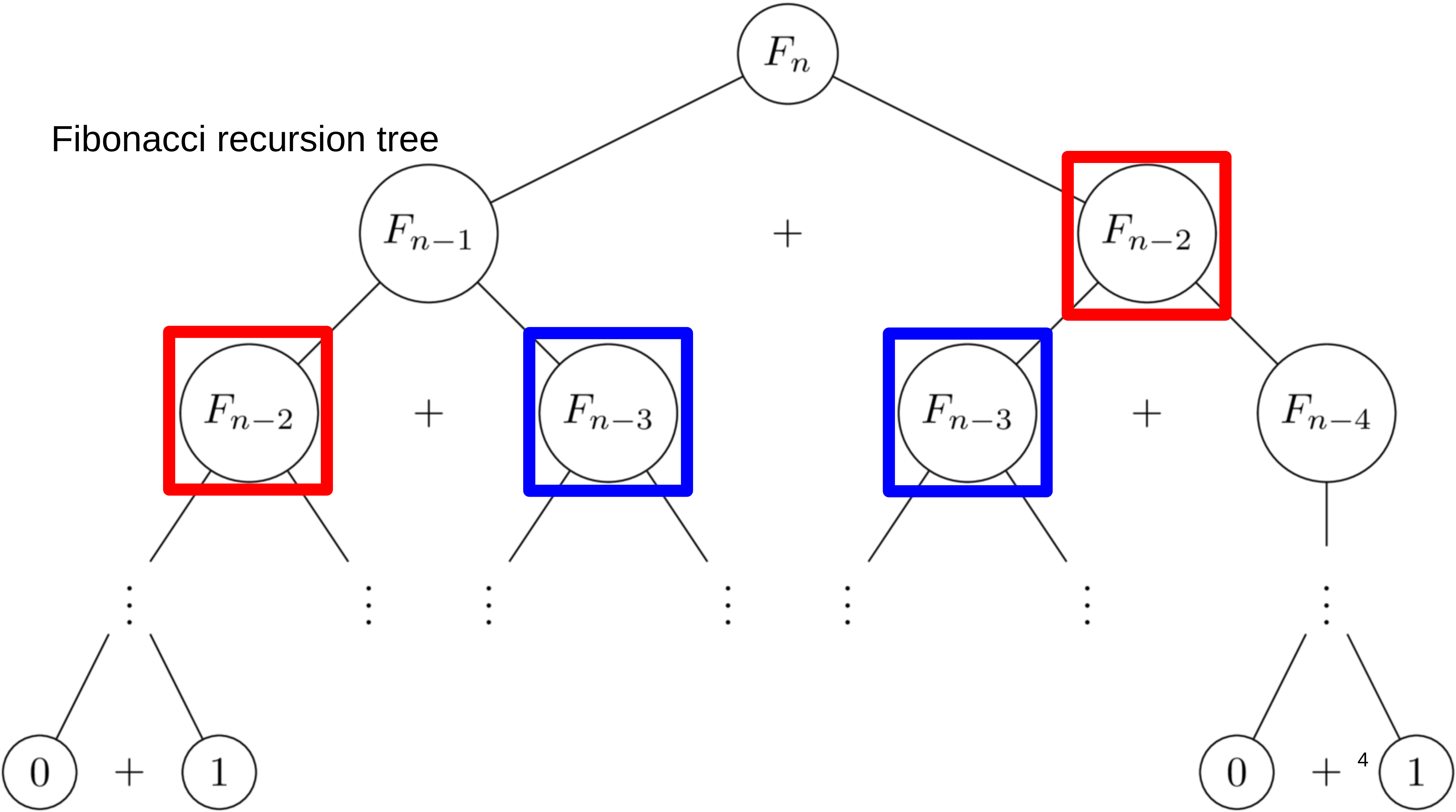
Main idea

Fibonacci recursion tree



Main idea

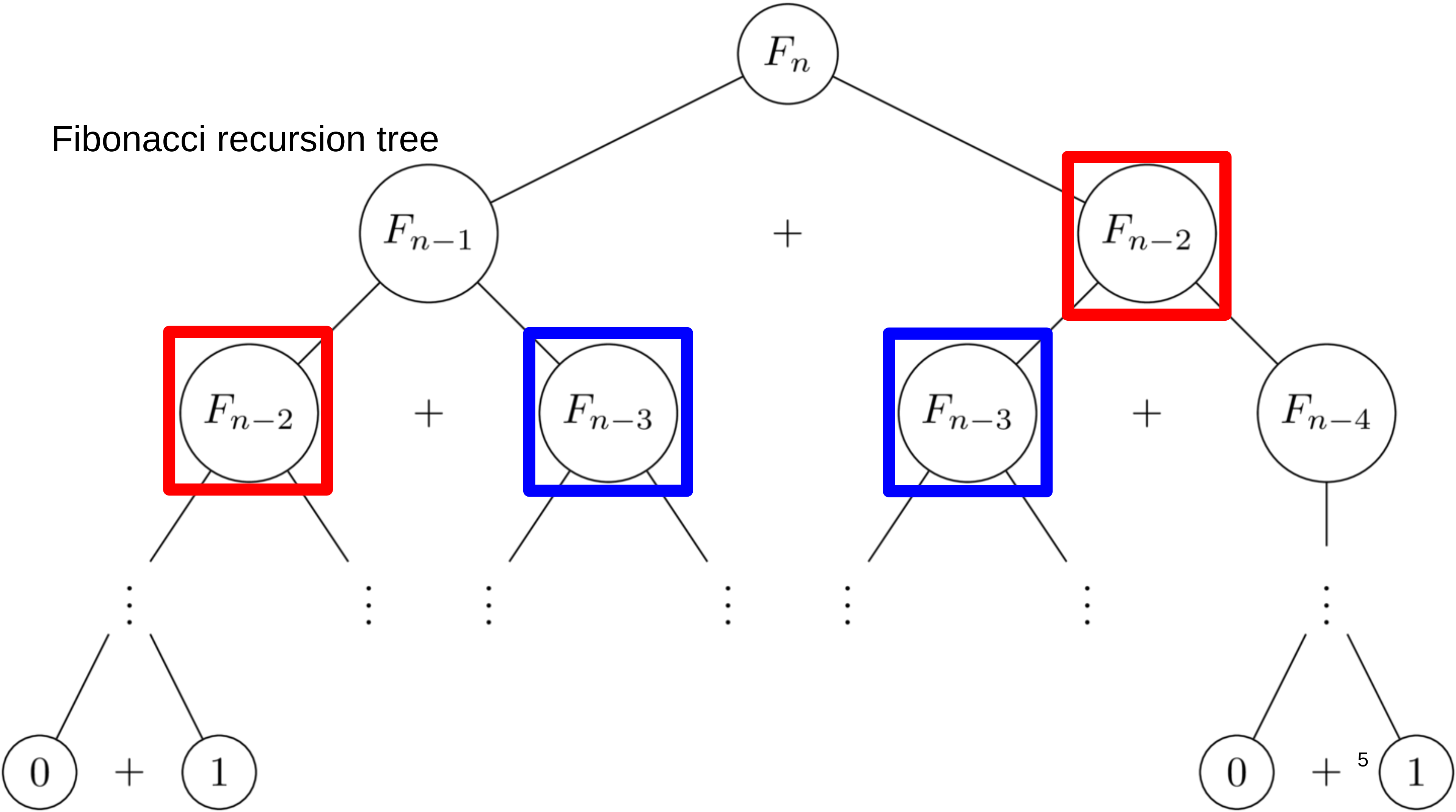
Fibonacci recursion tree



Main idea

- Do not repeat same work, store the result and look it up later

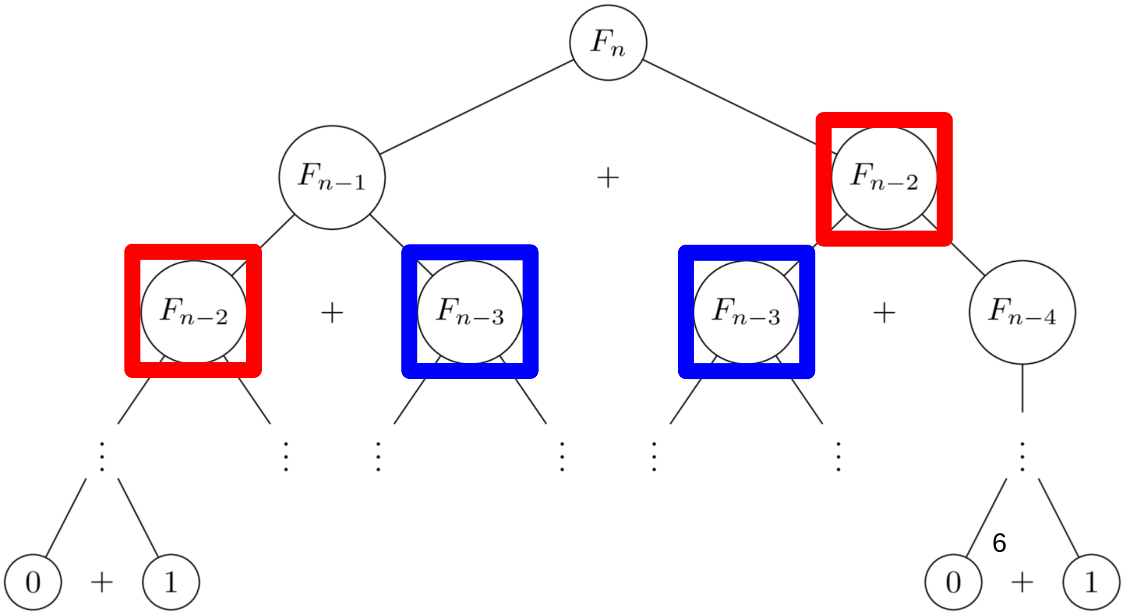
Fibonacci recursion tree



Main idea: DP vs Divide & Conquer

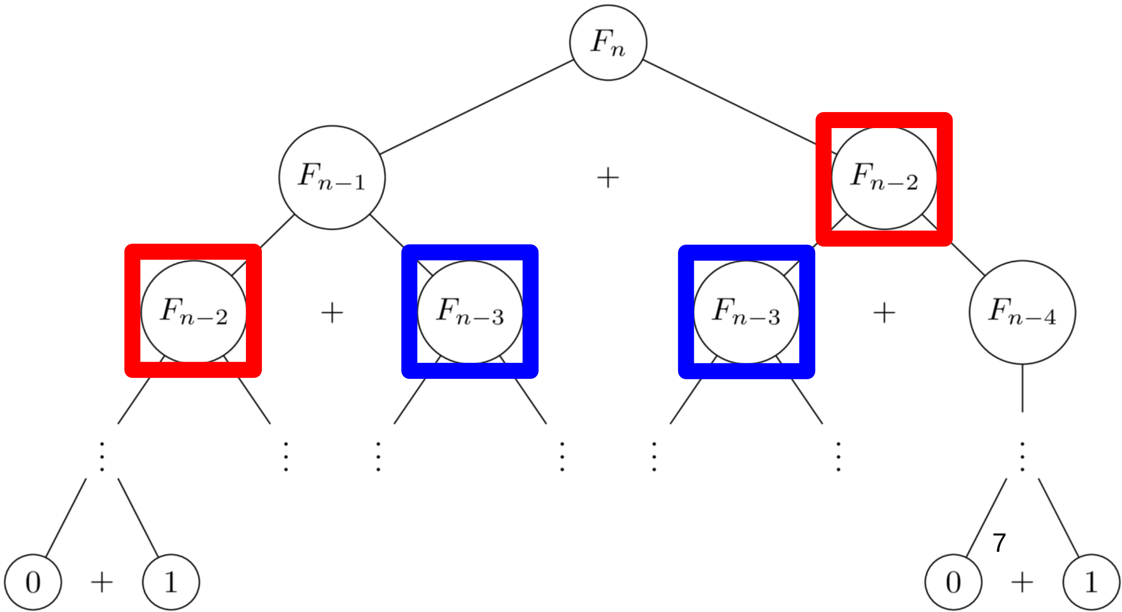
- Do not repeat same work, store the result and look it up later
- Is MergeSort(A, 1, n/2) and MergeSort(A, n/2, n) the same?

- Is Fib(n-2) and Fib(n-2) the same?



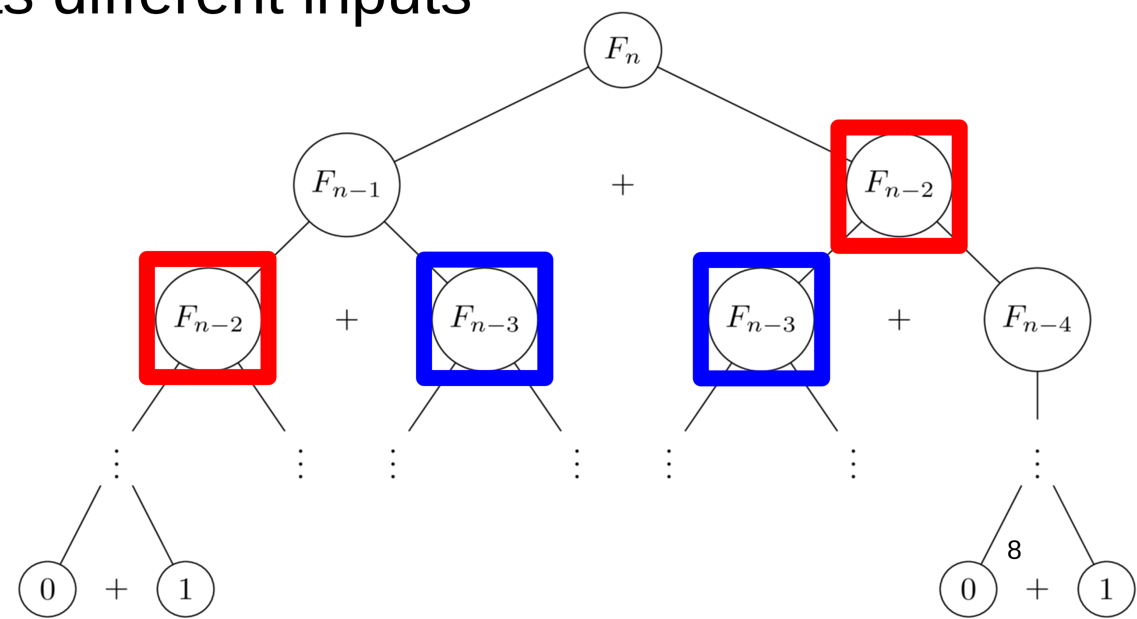
Main idea: DP vs Divide & Conquer

- Do not repeat same work, store the result and look it up later
- Is MergeSort(A, 1, n/2) and MergeSort(A, n/2, n) the same?
  - No
- Is Fib(n-2) and Fib(n-2) the same?
  - Yes



Main idea: DP vs Divide & Conquer

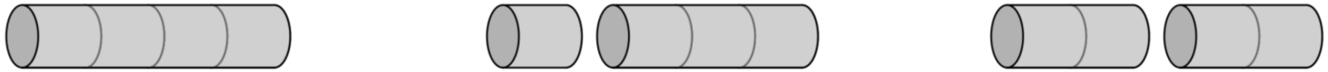
- Do not repeat same work, store the result and look it up later
- Is MergeSort(A, 1, n/2) and MergeSort(A, n/2, n) the same?
  - No
- Is Fib(n-2) and Fib(n-2) the same?
  - Yes
- Same function + same input → same output (DP)
- DC same function has different inputs



Rod Cutting Problem

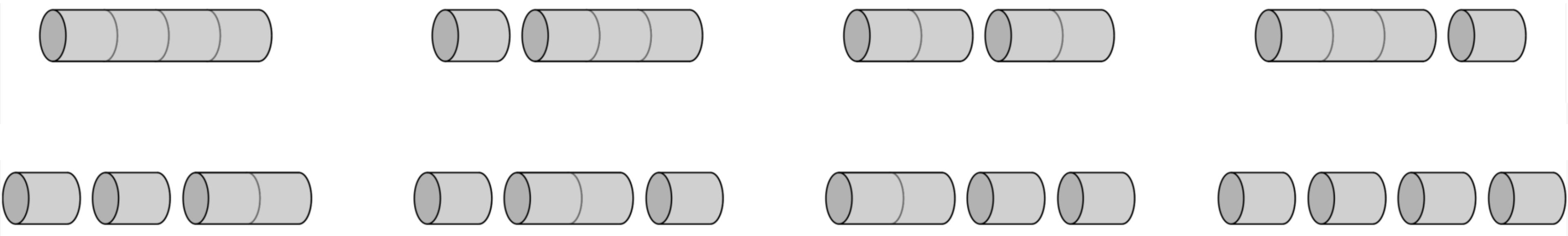
Rod Cutting Problem

Rod Cutting Problem

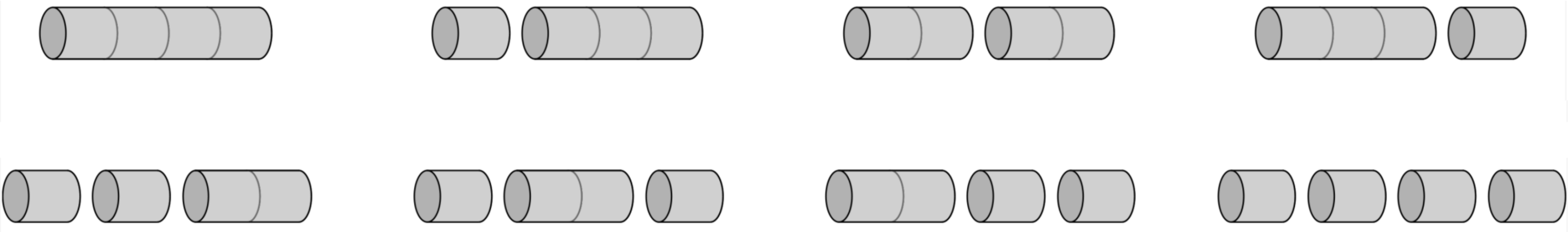


Rod Cutting Problem

Rod Cutting Problem

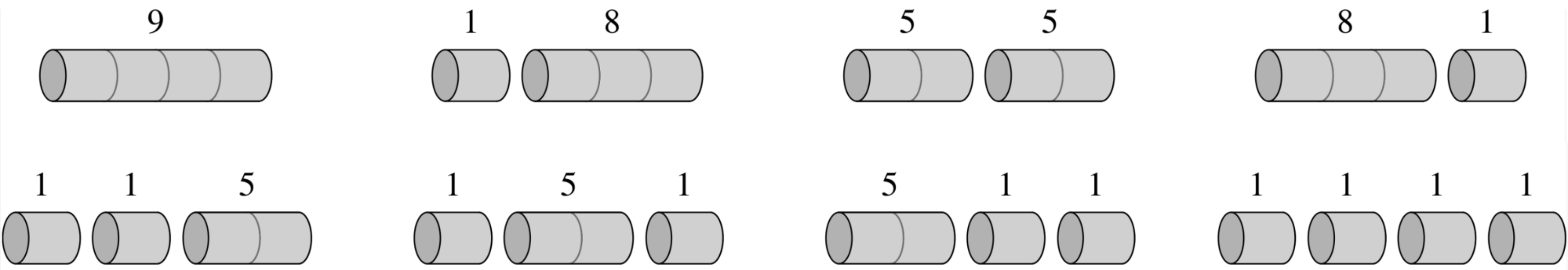


Rod Cutting Problem



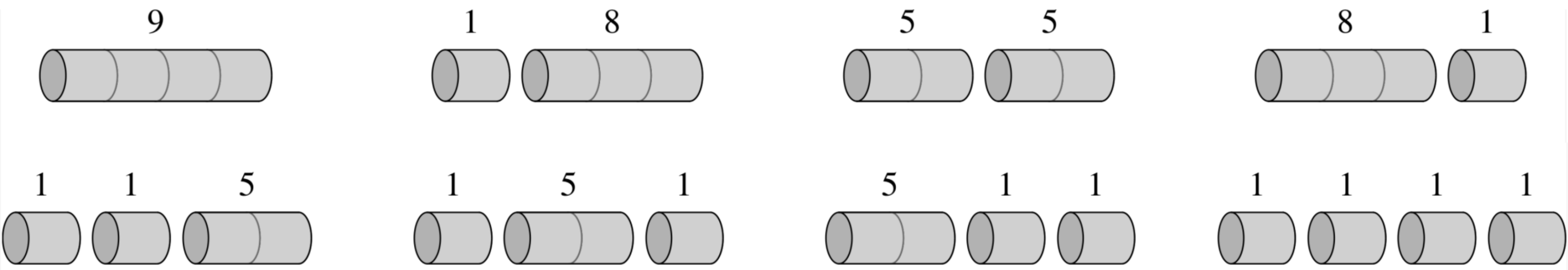
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod Cutting Problem



length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod Cutting Problem



length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Given a rod of length n and prices p_i, find the cutting strategy that makes the maximum revenue
  - In the example: (2+2) cutting makes r=5+5=10

Rod Cutting Problem

- Naïve: try all combinations

Rod Cutting Problem

- Naïve: try all combinations
  - How many?

Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: $^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: $^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: $^{n-1}C_{n-1} = \Theta(1)$

Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots.+n^3+n^2+n+1)$
    - Total: $O(n^n)$

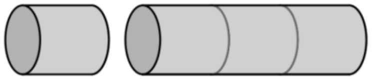
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots.+n^3+n^2+n+1)$
    - Total: $O(n^n)$
- Better solution? Can I divide and conquer?

Rod Cutting Problem

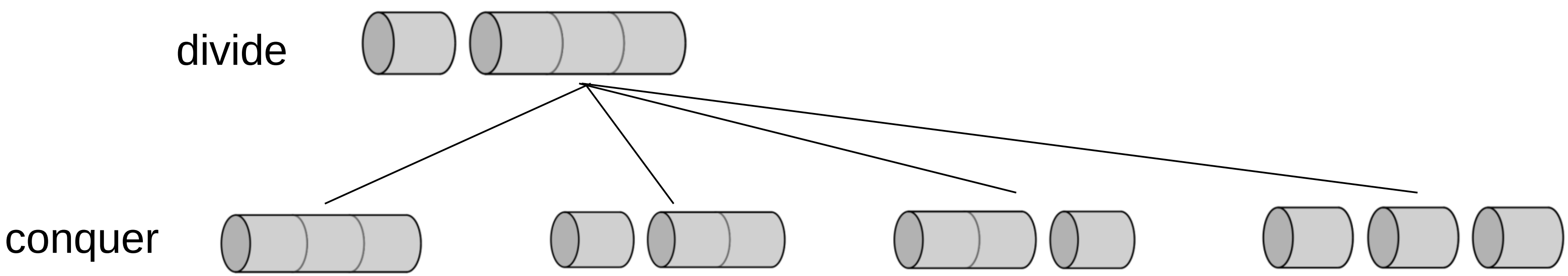
- Naïve: try all combinations
  - How many?
    - 0 cut: 1   1 cut: $(n-1)$ 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots.+n^3+n^2+n+1)$
    - Total: $O(n^n)$
- Better solution? Can I divide and conquer?

divide



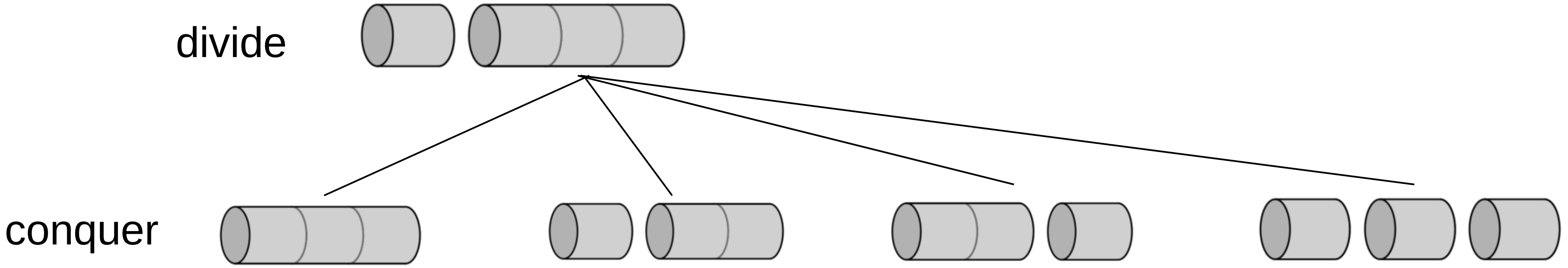
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots.+n^3+n^2+n+1)$
    - Total: $O(n^n)$
- Better solution? Can I divide and conquer?



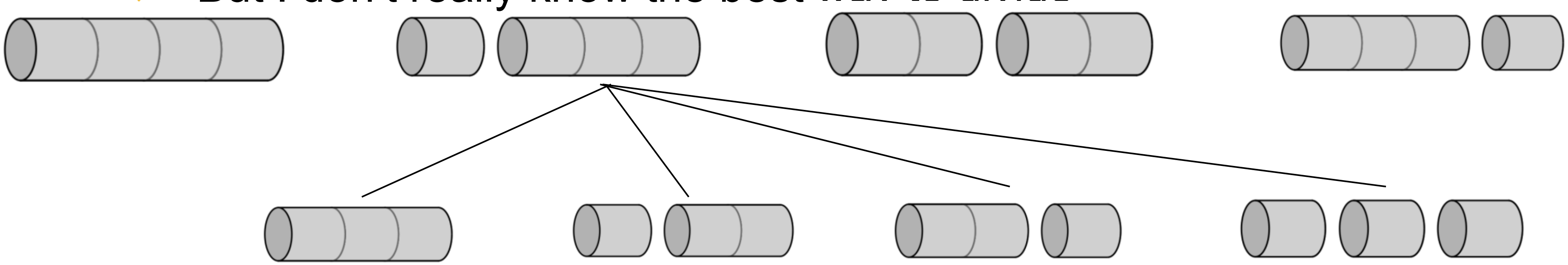
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1)$
    - Total: $O(n^n)$
- Better solution? Can I divide and conquer?
  - But I don't really know the best way to divide



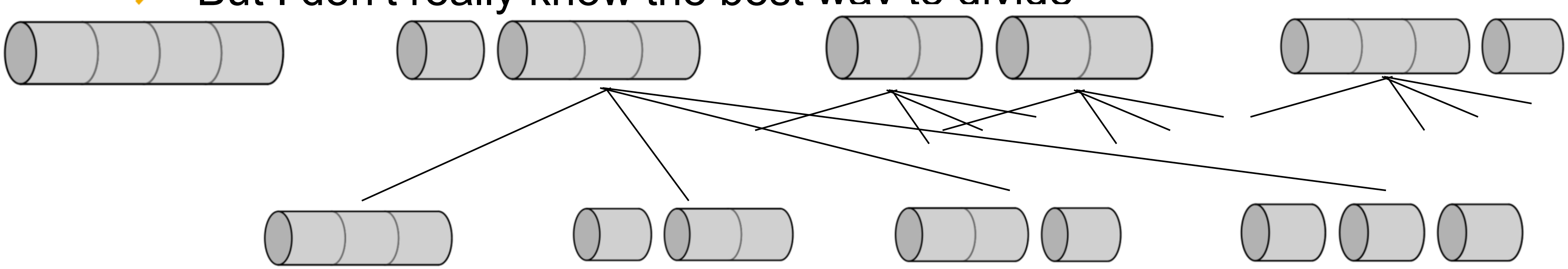
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots.+n^3+n^2+n+1)$
    - Total: $O(n^n)$
- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide



Rod Cutting Problem

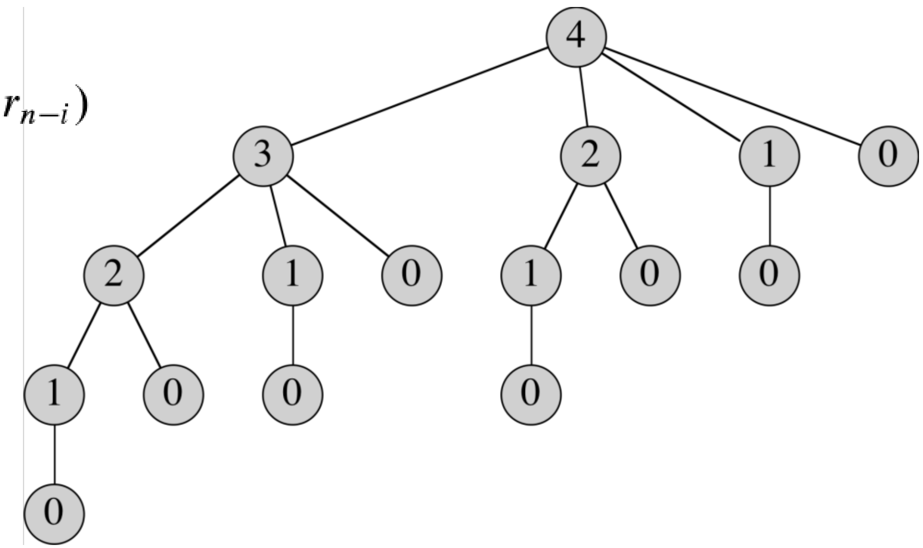
- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots.+n^3+n^2+n+1)$
    - Total: $O(n^n)$
- Better solution? Can I divide and conquer?
  - But I don't really know the best way to divide



Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1 1 cut: (n-1) 2 cuts: ${}^{n-1}C_2 = \Theta(n^2)$
    - 3 cuts: ${}^{n-1}C_3 = \Theta(n^3)$ …………… n cuts: ${}^{n-1}C_{n-1} = \Theta(1)$
    - Total: $\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1)$
    - Total: $O(n^n)$
- Better solution? Can I divide and conquer?
  - But I don't really know the best way to divide

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



Rod Cutting Problem

- Recursive top-down algorithm

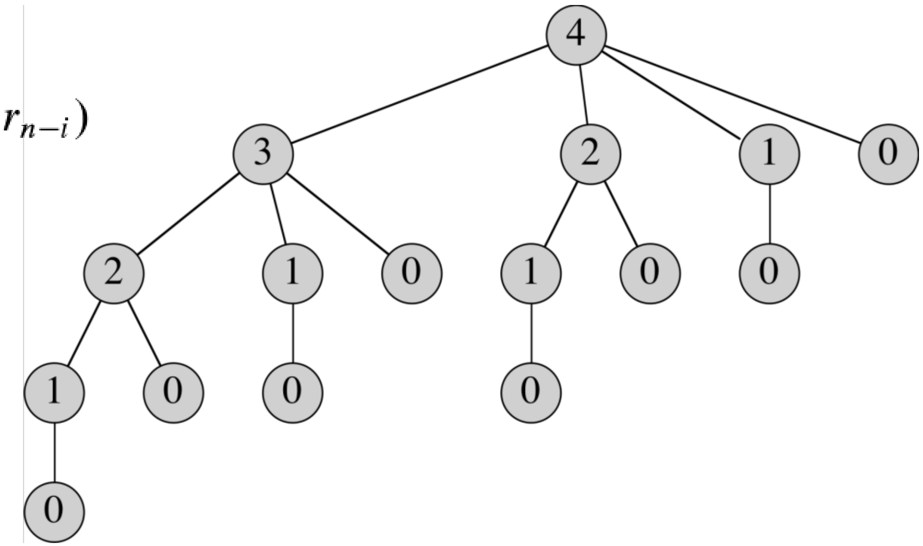
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

Rod Cutting Problem

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

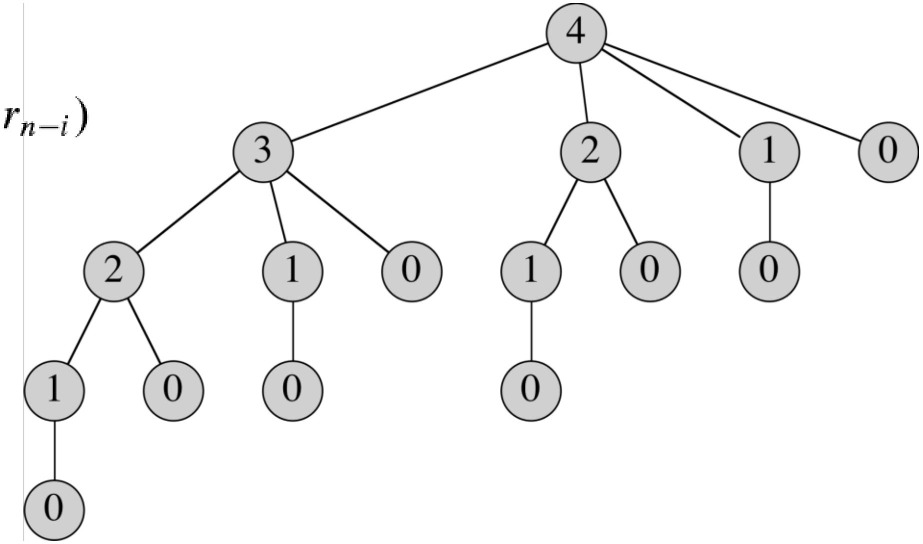


- How many subproblems (recursive calls)?

Rod Cutting Problem

- Better solution? Can I divide and conquer?
  - But I don't really know the best way to divide

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



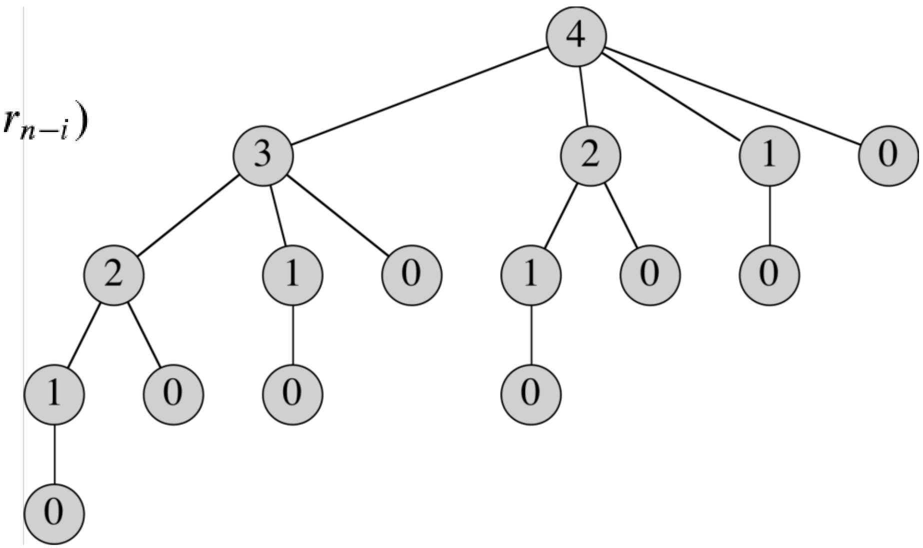
- How many subproblems (recursive calls)?

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) .$$

Rod Cutting Problem

- Better solution? Can I divide and conquer?
  - But I don't really know the best way to divide

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



- How many subproblems (recursive calls)?

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) .$$

$$T(n) = 2^n$$

(Prove by induction)

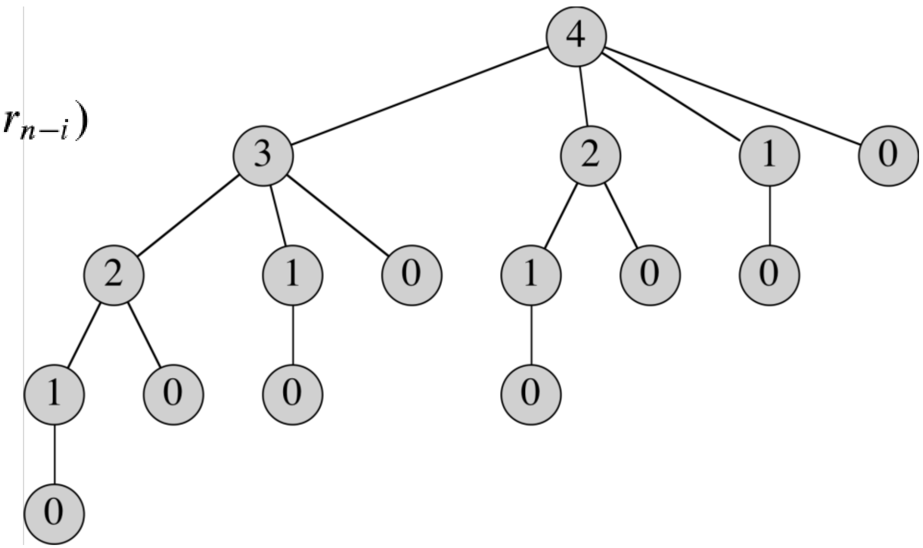
Rod Cutting Recursive Complexity

- Find the complexity of $\mathrm{T(n)} = 1 + \sum_{j=0}^{n-1} T(j)$
- Proof by induction:
  - Assume the solution is some function X(n)
  - Show that X(n) is true for the smallest n (the base case), e.g., n=0
  - Prove that X(n+1) is a solution for T(n+1) given X(n)
  - You are done
- Given $\mathrm{T(n)} = 1 + \sum_{j=0}^{n-1} T(j)$
- Assume $\mathrm{T(n)} = 2^n$
- $\mathrm{T(0)} = 1 + \sum_{j=0}^{-1} T(j) = 1 = 2^0$ (base case)
- $\mathrm{T(n+1)} = 1 + \sum_{j=0}^{n} T(j) = 1 + \sum_{j=0}^{n-1} T(j) + T(n) = T(n) + T(n) = 2T(n) = 2 * 2^n = 2^{n+1}$
- Then, $\mathrm{T(n)} = 2^n$

Rod Cutting Problem

- Better solution? Can I divide and conquer?
  - But I don't really know the best way to divide

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



- How many subproblems (recursive calls)?

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) .$$

$$T(n) = 2^n$$

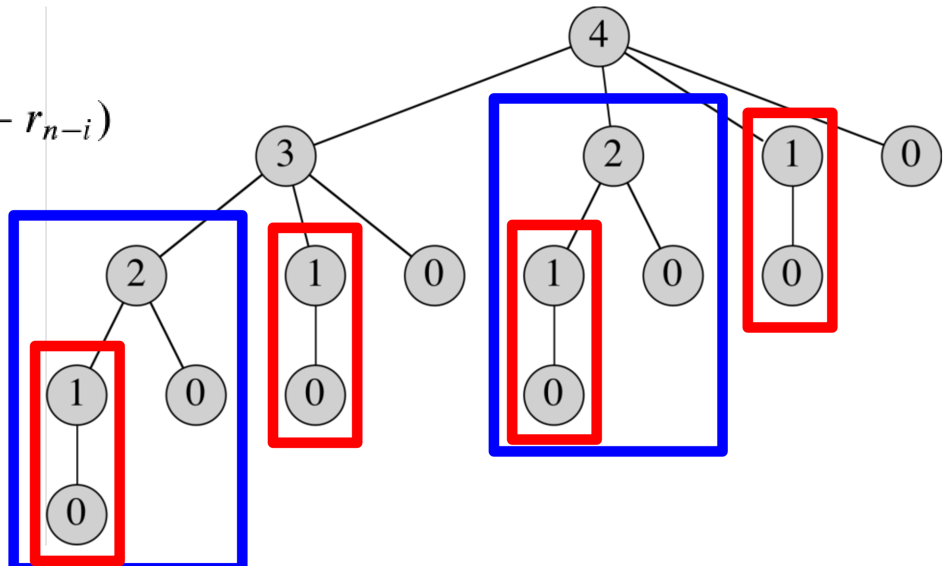
(Prove by induction)

- Can we do better?

Rod Cutting Problem

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



- How many subproblems (recursive calls)?

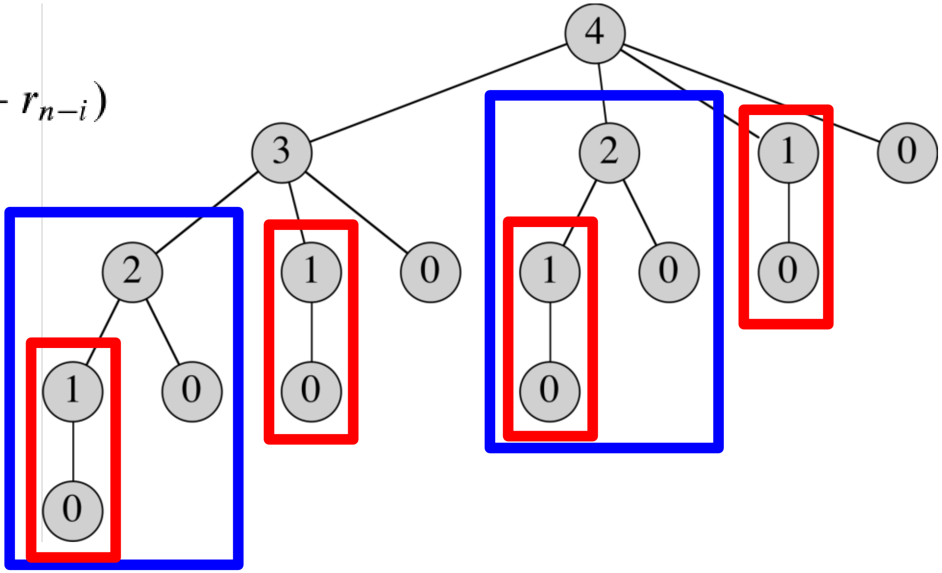
$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) .$$

$$T(n) = 2^n$$

- Can we do better?

Rod Cutting Problem

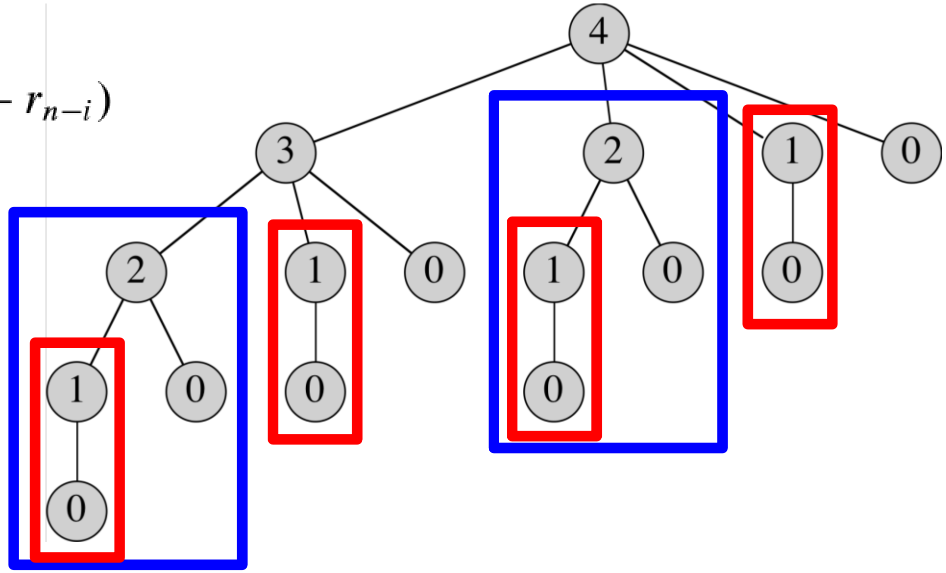
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



- Subproblem overlapping
  - No need to re-solve the same problem

Rod Cutting Problem

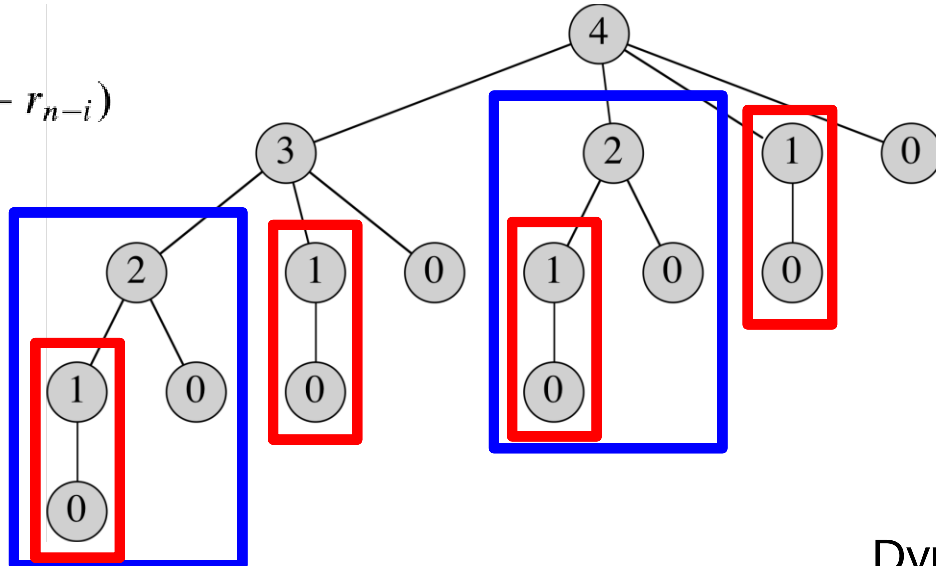
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



- Subproblem overlapping
  - No need to re-solve the same problem
- Idea:
  - Solve each subproblem once
  - Write down the solution in a lookup table (array, hashtable,…etc)
  - When needed again, look it up in Θ(1)

Rod Cutting Problem

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



Dynamic Programming

- Subproblem overlapping
  - No need to re-solve the same problem
- Idea:
  - Solve each subproblem once
  - Write down the solution in a lookup table (array, hashtable,…etc)
  - When needed again, look it up in $\Theta(1)$

Rod Cutting Problem

- Recursive top-down dynamic programming algorithm

```
MEMOIZED-CUT-ROD(p, n)
let r[0..n] be a new array
for i = 0 to n
    r[i] = −∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = −∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
r[n] = q
return q
```

Rod Cutting Problem

- Recursive top-down dynamic programming algorithm

```
MEMOIZED-CUT-ROD(p, n)
let r[0..n] be a new array
for i = 0 to n
    r[i] = -∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = -∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
r[n] = q
return q
```

$\Theta(n^2)$

Rod Cutting Problem

- Bottom-up dynamic programming algorithm
  - I know I will need the smaller problems → solve them first
  - Solve problem of size 0, then 1, then 2, then 3, … then n

Rod Cutting Problem

- Bottom-up dynamic programming algorithm
  - I know I will need the smaller problems → solve them first
  - Solve problem of size 0, then 1, then 2, then 3, … then n

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

Rod Cutting Problem

- Bottom-up dynamic programming algorithm
  - I know I will need the smaller problems → solve them first
  - Solve problem of size 0, then 1, then 2, then 3, … then n

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        q = max(q, p[i] + r[j − i])
    r[j] = q
return r[n]
```

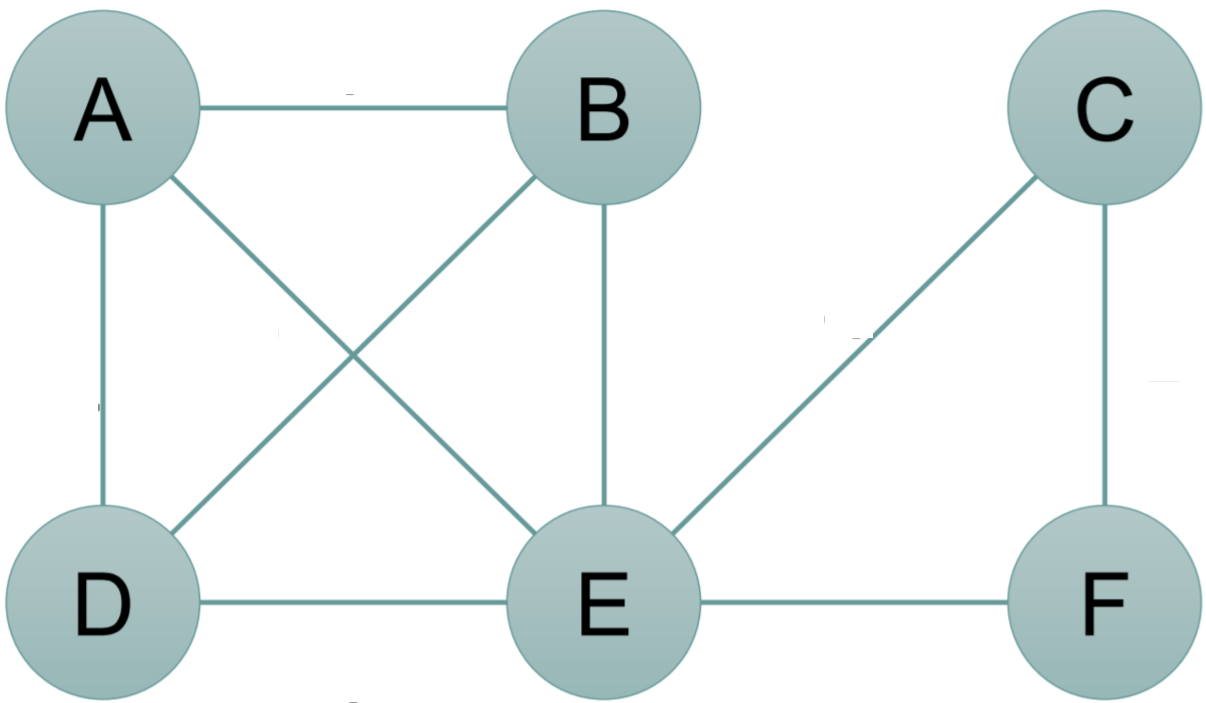
$\Theta(n^2)$

Elements of a Dynamic Programming Problem

- Optimal substructure
  - Optimal solution of a larger problem comes from optimal solutions of smaller problems
- Subproblem overlapping
  - Same exact sub-problems are solved again and again

Optimal Substructure

- Longest path from A to F LP(A,F) includes node B
  - But it does not include LP(A,B) and LP(B,F)
  - i.e., optimal solutions for the subproblems A→B, and B→F cannot be combined to find an optimal solution for A → F



Dynamic Programming vs. D&C

- How different?

Dynamic Programming vs. D&C

- How different?
  - No subproblem overlapping
    - Each subproblem with distinct input is a new problem
  - Not necessarily optimization problems, i.e., no objective function

Reconstructing Solution

- Rod cutting problem: What are the actual cuts?
  - Not only the best revenue (the optimal objective function value)

Reconstructing Solution

- Rod cutting problem: What are the actual cuts?
  - Not only the best revenue (the optimal objective function value)

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[1..n] be new arrays
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        if q < p[i] + r[j − i]
            q = p[i] + r[j − i]
            s[j] = i
    r[j] = q
return r and s
```

Reconstructing Solution

- Rod cutting problem: What are the actual cuts?
  - Not only the best revenue (the optimal objective function value)

```
PRINT-CUT-ROD-SOLUTION(p, n)
(r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
while n > 0
    print s[n]
    n = n - s[n]
```

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$		1	2	3	2	2	6	1	2	3	10

  - Let’s trace examples

Matrix Chain Multiplication

- How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?

Matrix Chain Multiplication

- How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?

$$(A_1(A_2(A_3A_4)))$$
$$(A_1((A_2A_3)A_4))$$
$$((A_1A_2)(A_3A_4))$$
$$((A_1(A_2A_3))A_4)$$
$$(((A_1A_2)A_3)A_4)$$

Matrix Chain Multiplication

- How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?

$$(A_1(A_2(A_3A_4)))$$
$$(A_1((A_2A_3)A_4))$$
$$((A_1A_2)(A_3A_4))$$
$$((A_1(A_2A_3))A_4)$$
$$(((A_1A_2)A_3)A_4)$$

- Does it really make a difference?

Matrix Chain Multiplication

- How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?

$$(A_1(A_2(A_3A_4)))$$
$$(A_1((A_2A_3)A_4))$$
$$((A_1A_2)(A_3A_4))$$
$$((A_1(A_2A_3))A_4)$$
$$(((A_1A_2)A_3)A_4)$$

- Does it really make a difference?
- # of multiplications:

A.rows*B.cols*A.cols

```
MATRIX-MULTIPLY(A, B)
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

Matrix Chain Multiplication

- Does it really make a difference?
- # of multiplications:

A.rows*B.cols*A.cols

- Example:

A1*A2*A3

Dimensions:

10x100x5x50

MATRIX-MULTIPLY(A, B)

```
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

- # of multiplications in ((A1*A2)*A3)=10*100*5+10*5*50=7.5K
- # of multiplications in (A1*(A2*A3))=100*5*50+10*100*50=75K

Matrix Chain Multiplication

- Given n matrices A_1 A_2 … A_n of dimensions p_0 p_1 … p_n, find the optimal parentheses to multiply the matrix chain

Matrix Chain Multiplication

- Given n matrices $A_1\ A_2 \ldots A_n$ of dimensions $p_0\ p_1 \ldots p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1\ A_2\ A_3\ A_4\ A_5 \ldots A_n$

Matrix Chain Multiplication

- Given n matrices A_1 A_2 … A_n of dimensions p_0 p_1 … p_n, find the optimal parentheses to multiply the matrix chain
- $A_1\ A_2\ A_3\ A_4\ A_5 \ldots A_n$
- $(A_1\ A_2\ A_3)(A_4\ A_5 \ldots A_n)$

Matrix Chain Multiplication

- Given n matrices $A_1\ A_2 \dots A_n$ of dimensions $p_0\ p_1 \dots p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1\ A_2\ A_3\ A_4\ A_5 \dots A_n$
- $(A_1\ A_2\ A_3)(A_4\ A_5 \dots A_n)$
- Sub-chains C1 = $(A_1\ A_2\ A_3)$, C2 = $(A_4\ A_5 \dots A_n)$

Matrix Chain Multiplication

- Given n matrices $A_1\ A_2 \ldots A_n$ of dimensions $p_0\ p_1 \ldots p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1\ A_2\ A_3\ A_4\ A_5 \ldots A_n$
- $(A_1\ A_2\ A_3)(A_4\ A_5 \ldots A_n)$
- Sub-chains C1 = $(A_1\ A_2\ A_3)$, C2 = $(A_4\ A_5 \ldots A_n)$
- Total Cost C = cost(C1)+cost(C2)+$p_0 p_3 p_n$

Matrix Chain Multiplication

- Given n matrices $A_1\ A_2 \ldots A_n$ of dimensions $p_0\ p_1 \ldots p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1\ A_2\ A_3\ A_4\ A_5 \ldots A_n$
- $(A_1\ A_2\ A_3)(A_4\ A_5 \ldots A_n)$
- Sub-chains C1 = $(A_1\ A_2\ A_3)$, C2 = $(A_4\ A_5 \ldots A_n)$
- Total Cost C = cost(C1)+cost(C2)+$p_0 p_3 p_n$
- Then, if cost(C1) and cost(C2) are minimal (i.e., optimal), then C is optimal (optimal substructure holds)

Matrix Chain Multiplication

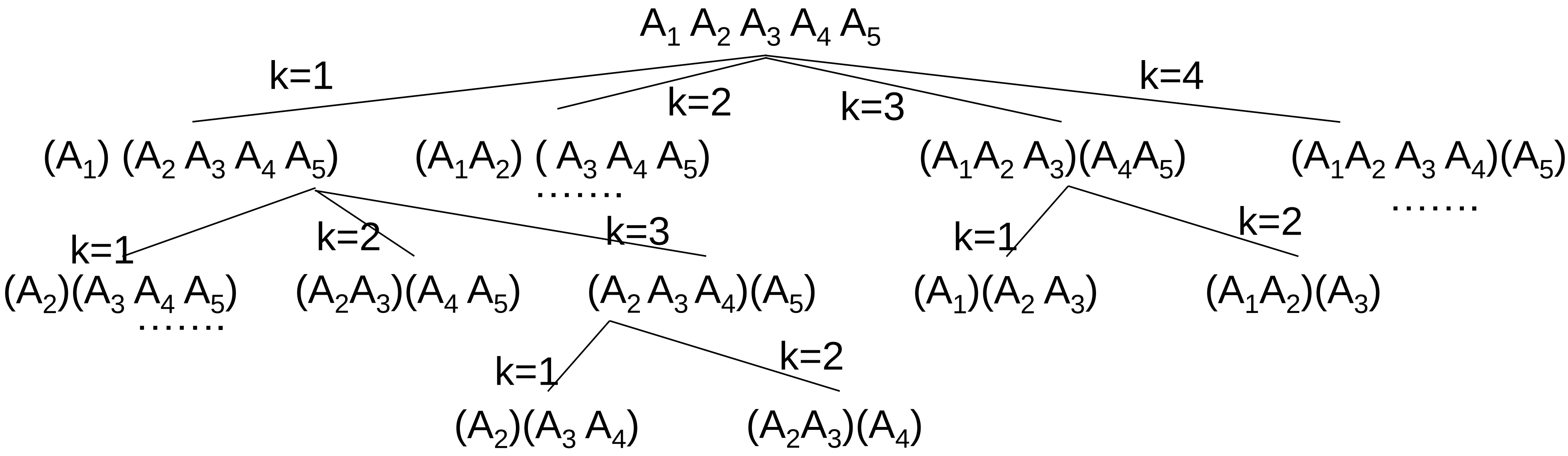
- Given n matrices $A_1\ A_2 \ldots A_n$ of dimensions $p_0\ p_1 \ldots p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1\ A_2\ A_3\ A_4\ A_5 \ldots A_n$
- $(A_1\ A_2\ A_3)(A_4\ A_5 \ldots A_n)$
- Sub-chains $C1 = (A_1\ A_2\ A_3)$, $C2 = (A_4\ A_5 \ldots A_n)$
- Total Cost $C = cost(C1)+cost(C2)+p_0p_3p_n$
- Then, if cost(C1) and cost(C2) are minimal (i.e., optimal), then C is optimal (optimal substructure holds)
- Proof by contradiction:
  - Given C is optimal, are cost(C1)=c1 and cost(C2)=c2 optimal?
  - Assume c1 is NOT optimal, then ∃ an optimal solution of cost c1' < c1
  - Then c1'+c2+p < c1+c2+p → C' < C
  - Then C is not optimal → contradiction!
  - Then C1 has to be optimal → optimal substructure holds

Matrix Chain Multiplication

- Given n matrices $A_1\ A_2\ \ldots\ A_n$ of dimensions $p_0\ p_1\ \ldots\ p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1\ A_2\ A_3\ A_4\ A_5\ \ldots\ A_n$
- $(A_1\ A_2\ A_3)(A_4\ A_5\ \ldots\ A_n)$
- Sub-chains $C1 = (A_1\ A_2\ A_3)$, $C2 = (A_4\ A_5\ \ldots\ A_n)$
- Total Cost $C = cost(C1)+cost(C2)+p_0p_3p_n$
- Then, if cost(C1) and cost(C2) are minimal (i.e., optimal), then C is optimal (optimal substructure holds)
- Optimal C1, C2 might be one of different options
  - $C1 = (A_1\ A_2)$, $C2 = (A_3\ A_4\ A_5\ \ldots\ A_n)$
  - $C1 = (A_1)$, $C2 = (A_2\ A_3\ A_4\ A_5\ \ldots\ A_n)$
  - $C1 = (A_1\ A_2\ A_3\ A_4)$, $C2 = (A_5\ \ldots\ A_n)$
  - …….

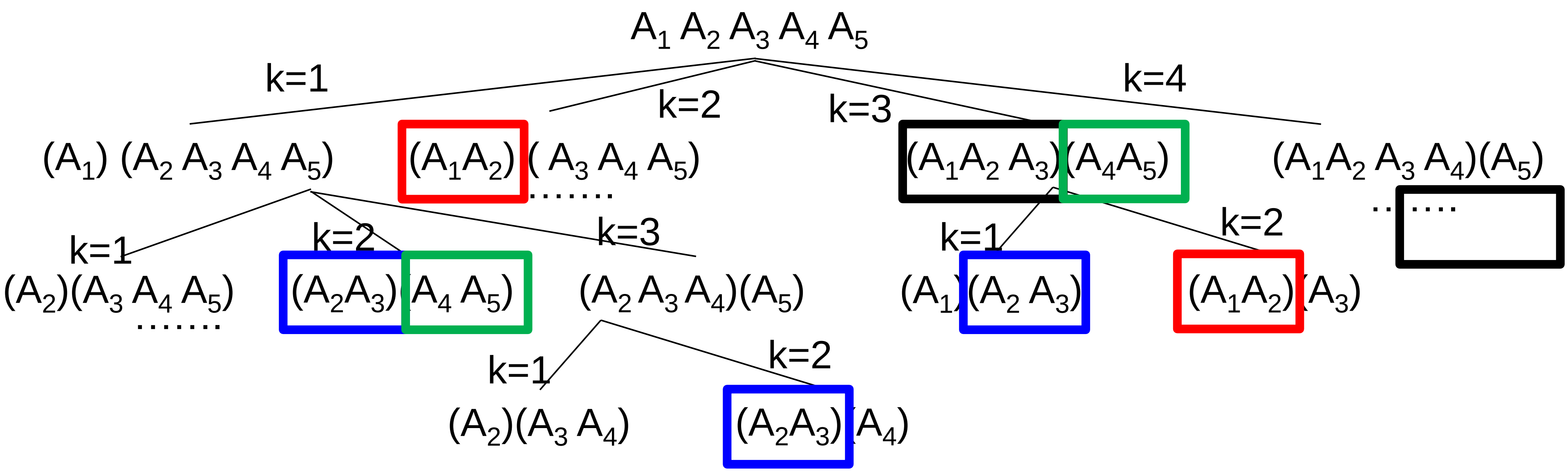
Matrix Chain Multiplication

- Assume k is length of first sub-chain C1



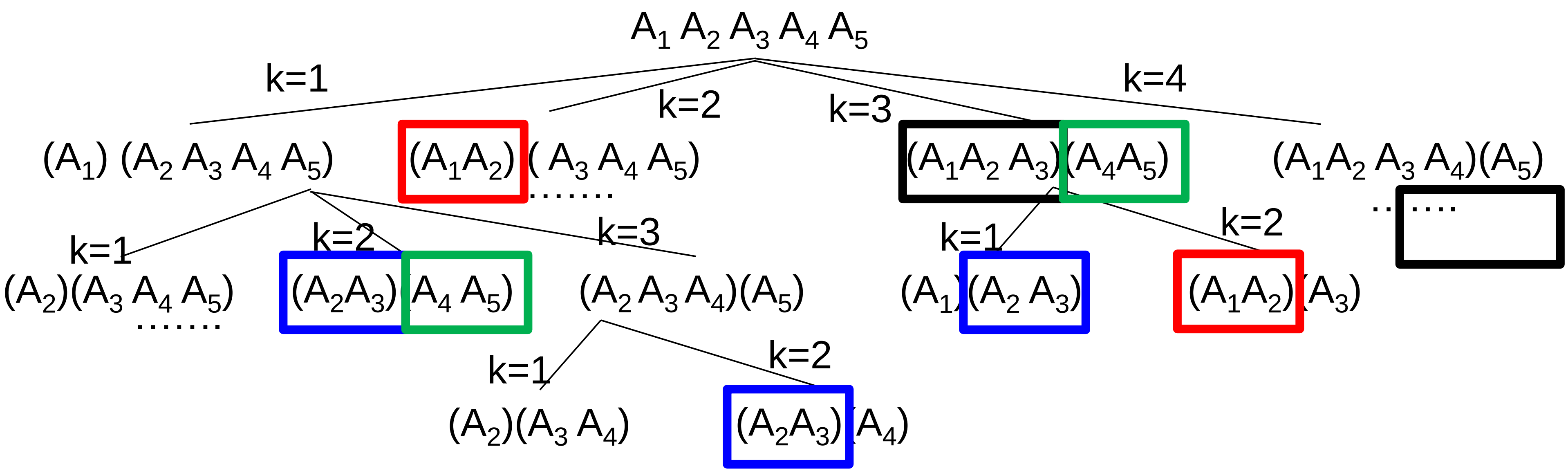
Matrix Chain Multiplication

- Assume k is length of first sub-chain C1



Matrix Chain Multiplication

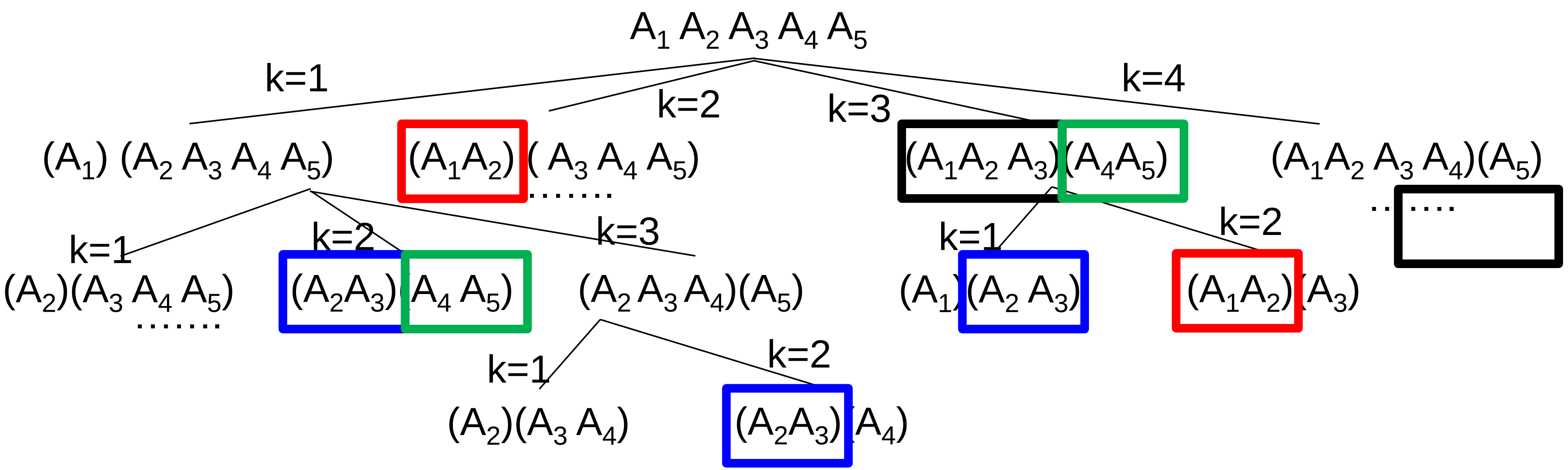
- Assume k is length of first sub-chain C1



- Obviously, a lot of overlapping subproblems appear

Matrix Chain Multiplication

- Assume k is length of first sub-chain C1



- Obviously, a lot of overlapping subproblems appear
- Optimal substructure + subproblem overlapping = dynamic programming

Matrix Chain Multiplication

- Generally: $A_i \ldots A_k \ldots A_j$ of dimensions $p_i \ldots p_k \ldots p_j$
- $(A_i \ldots A_k)(A_{k+1} \ldots A_j)$, where k=i,i+1,…j-1
- Then, solve each sub-chains recursively

Matrix Chain Multiplication

- Generally: $A_i \ldots A_k \ldots A_j$ of dimensions $p_i \ldots p_k \ldots p_j$
- $(A_i \ldots A_k)(A_{k+1} \ldots A_j)$, where k=i,i+1,…j-1
- Then, solve each sub-chains recursively

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

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?

Matrix Chain Multiplication: Designing Algorithm

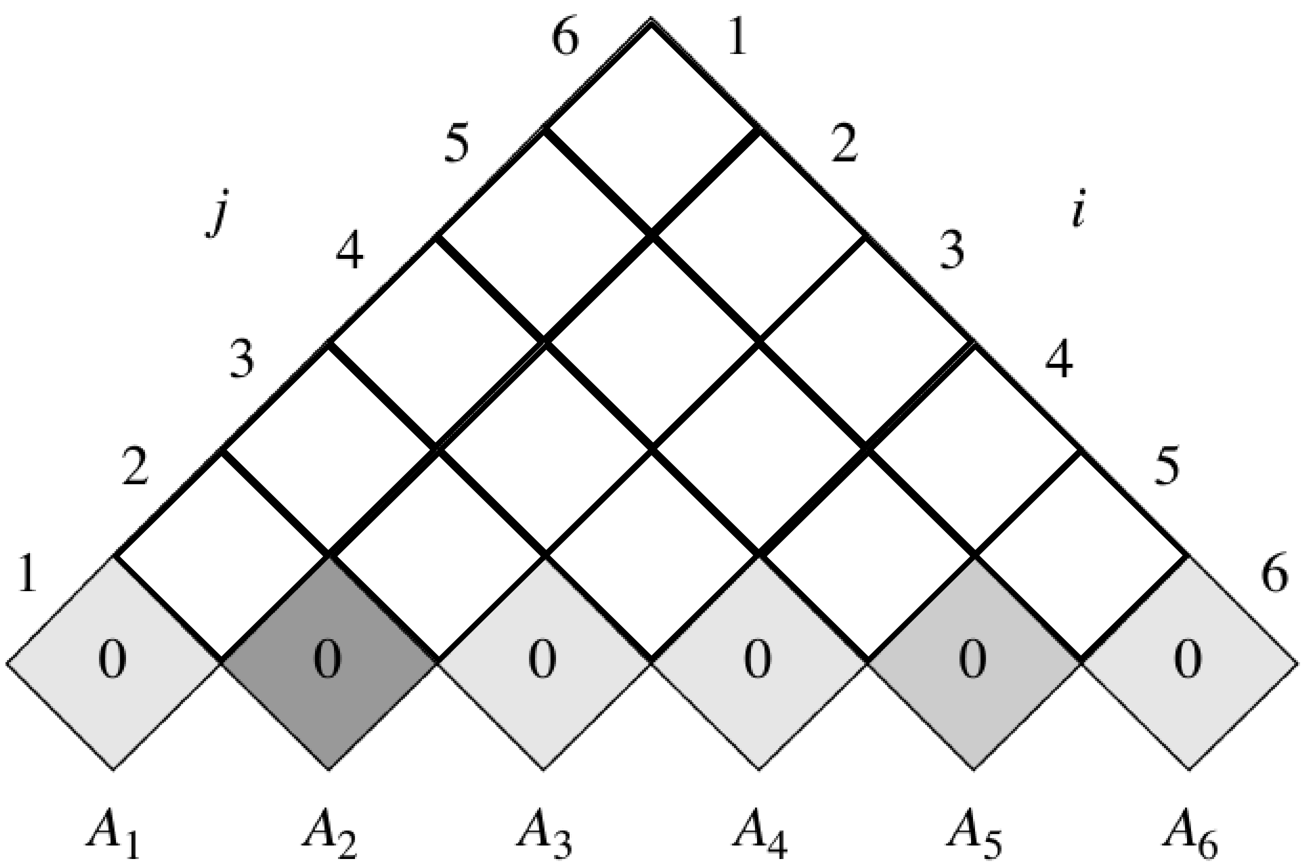
- What is the smallest subproblem?
  - A chain of length 2

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

Matrix Chain Multiplication: Designing Algorithm

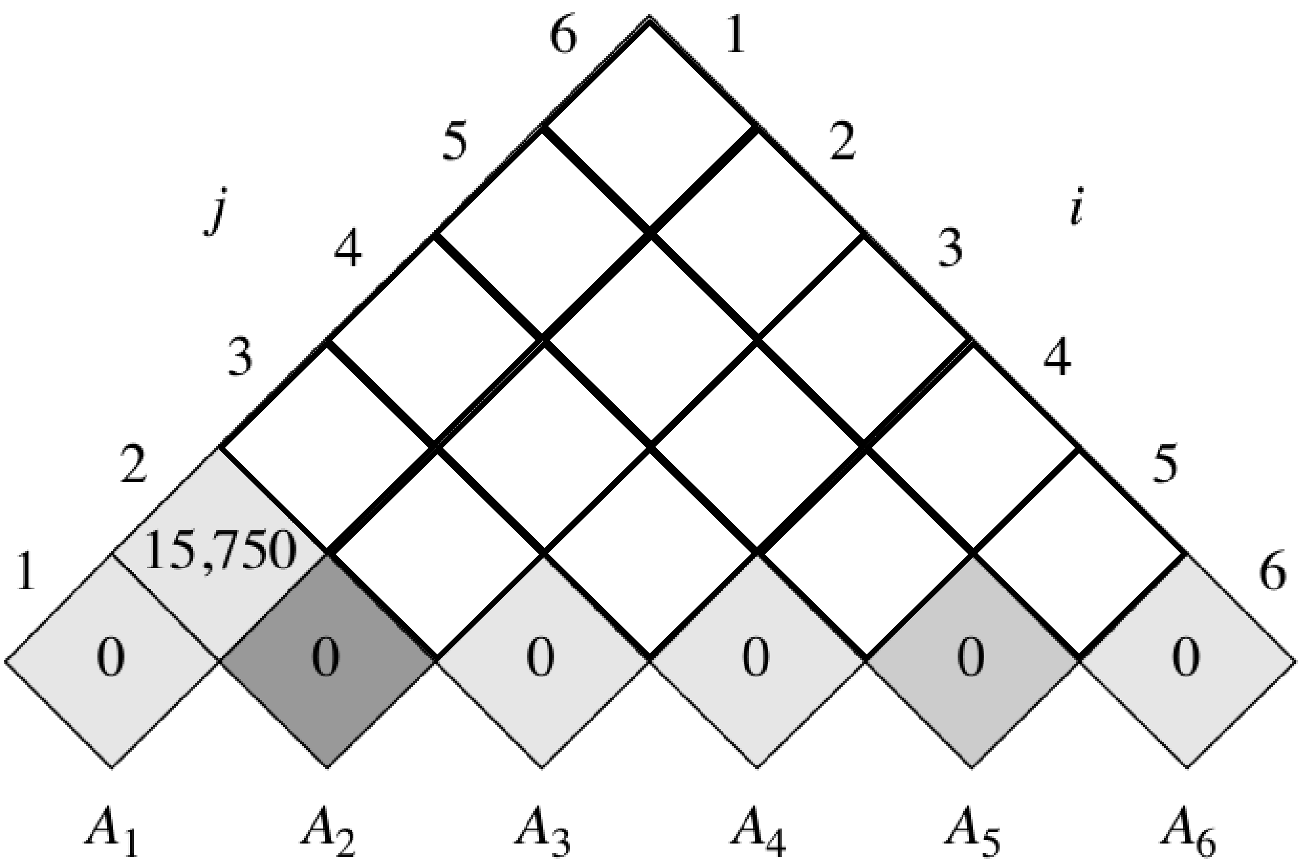
- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n



A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

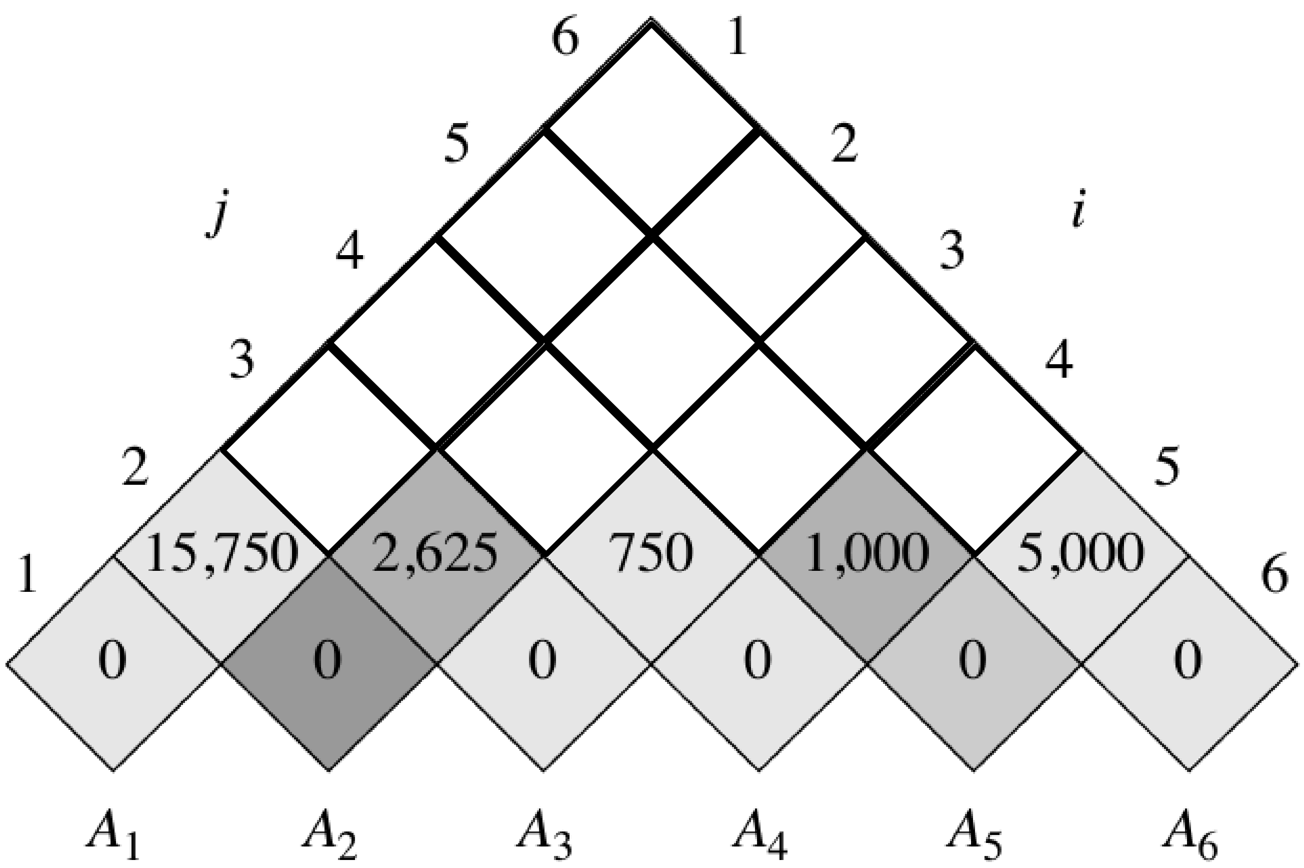
- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n



A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

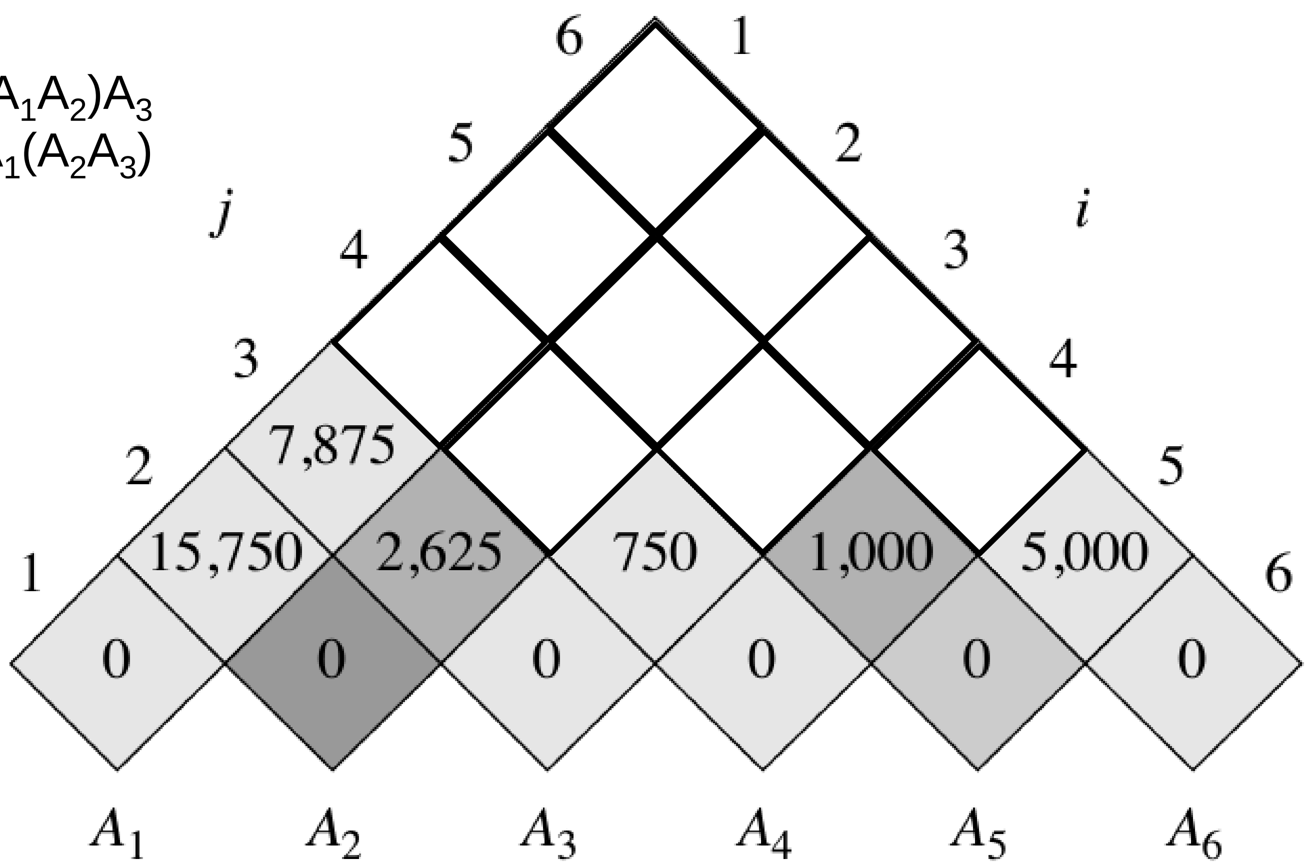


A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

$A_1A_2A_3 = (A_1A_2)A_3$
Or $A_1(A_2A_3)$

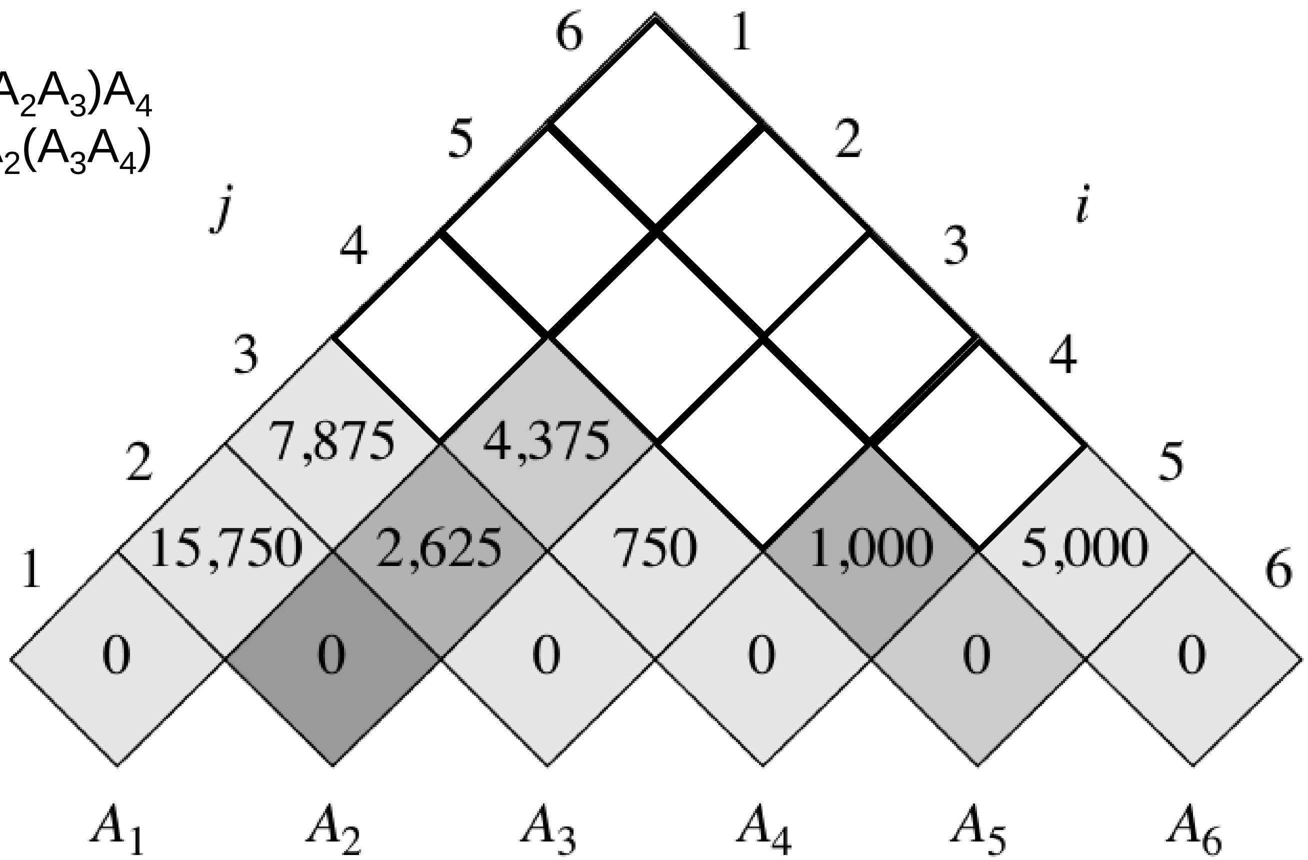


A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

$A_2A_3A_4 = (A_2A_3)A_4$
Or $A_2(A_3A_4)$

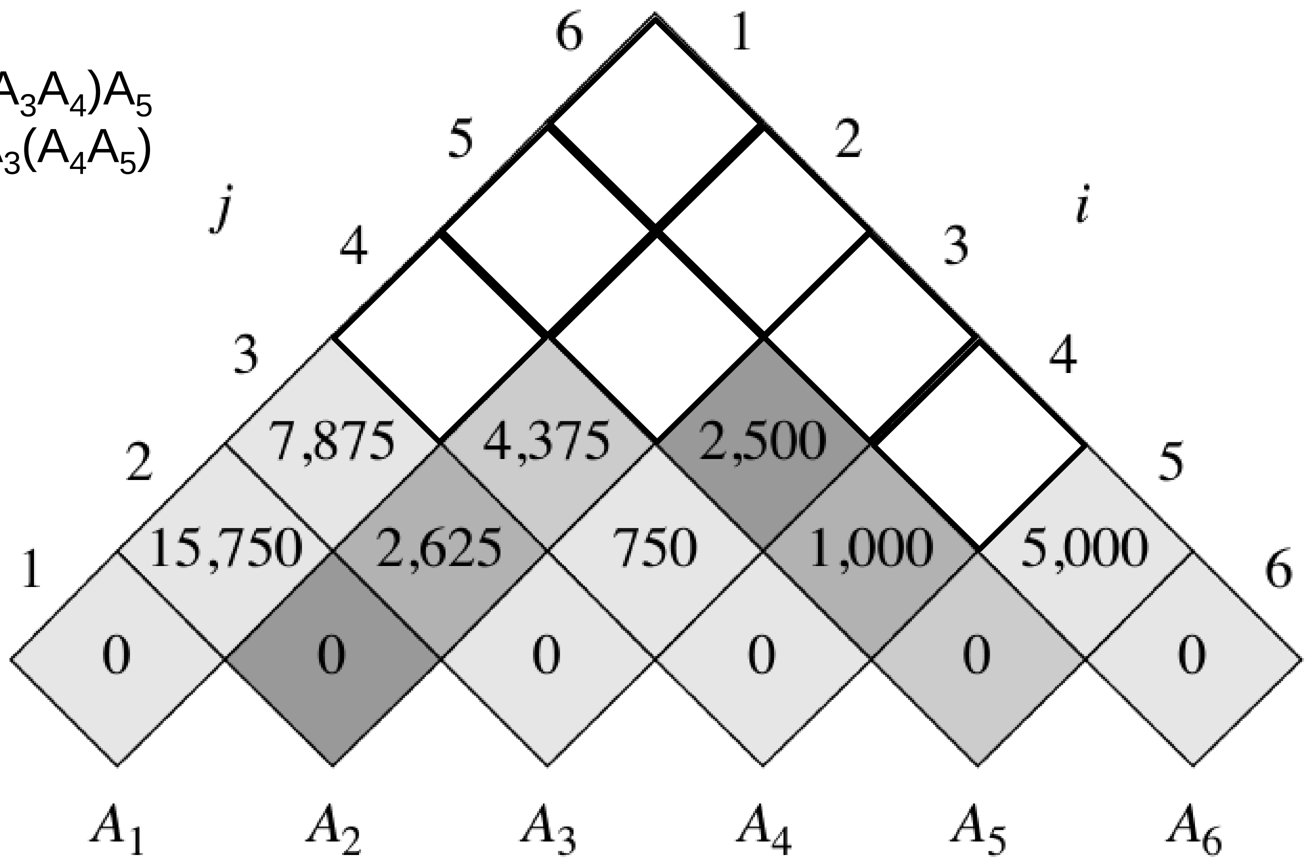


A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

$A_3A_4A_5 = (A_3A_4)A_5$
Or $A_3(A_4A_5)$

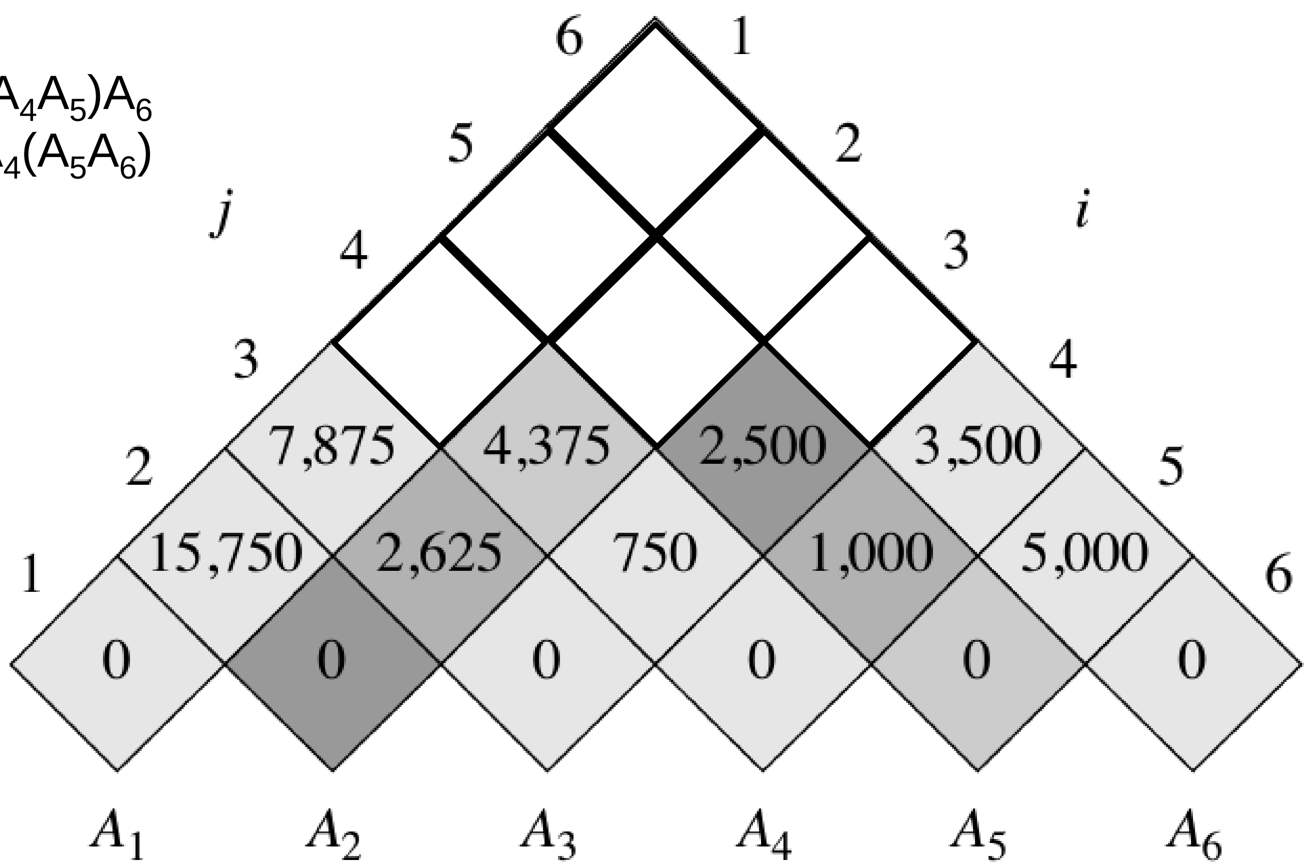


A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

$A_4A_5A_6 = (A_4A_5)A_6$
Or $A_4(A_5A_6)$

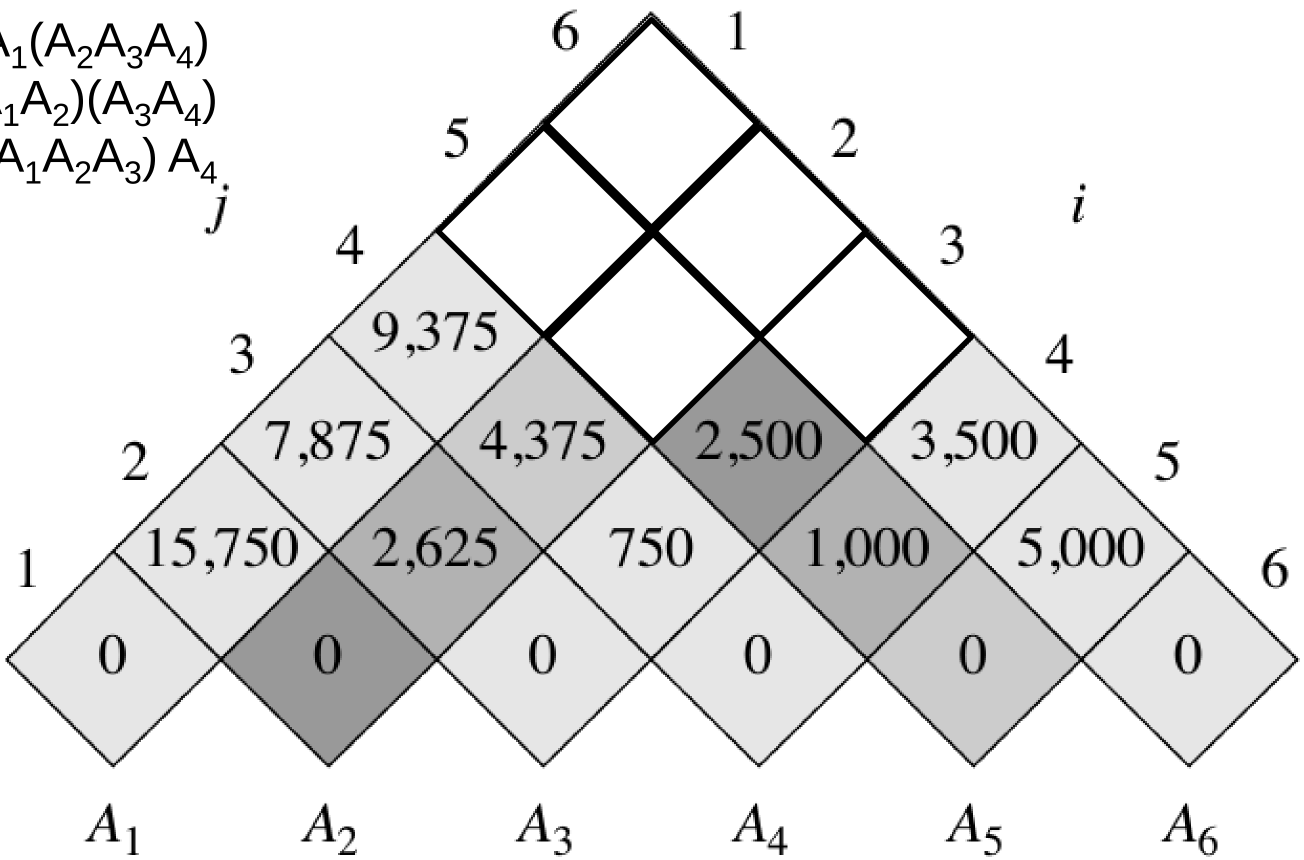


A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

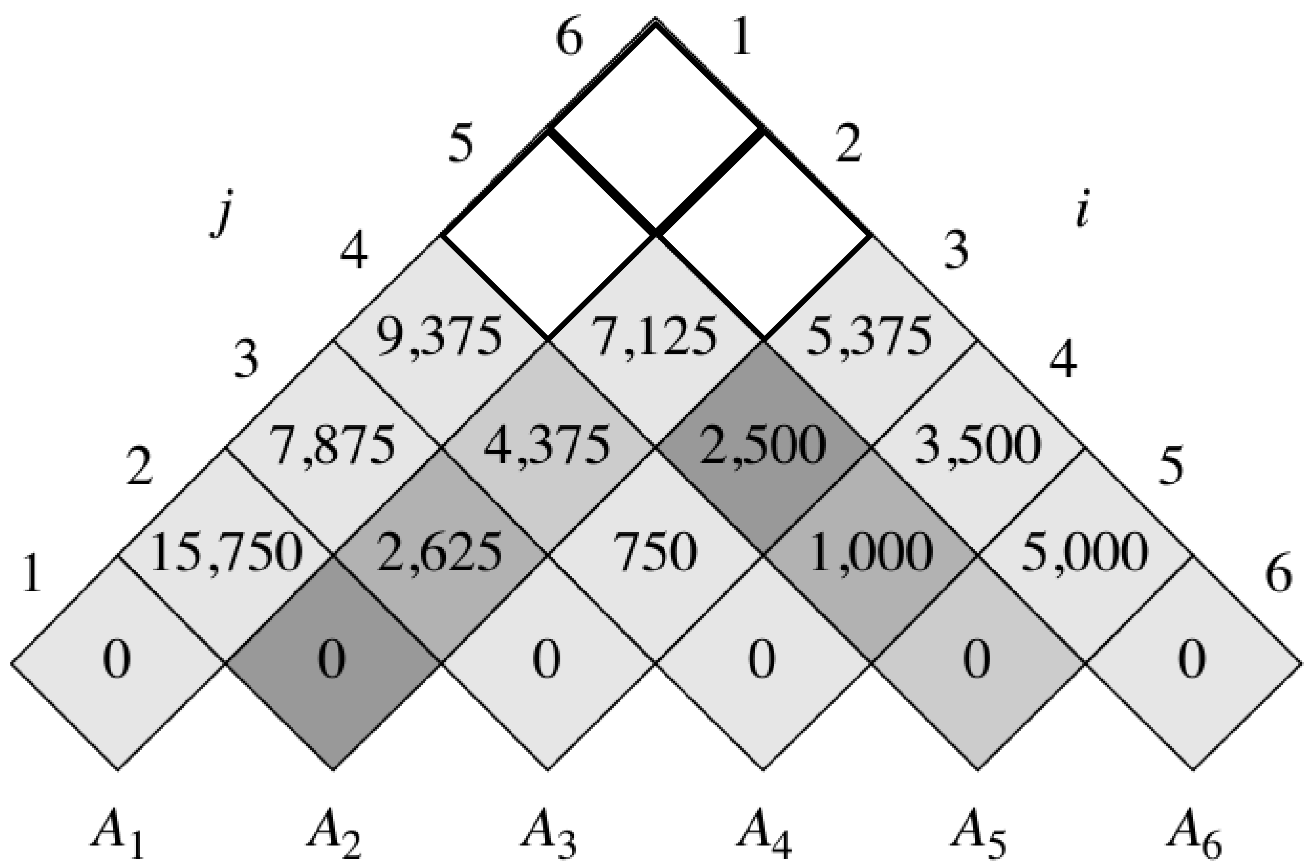
$A_1A_2A_3A_4 = A_1(A_2A_3A_4)$
Or $(A_1A_2)(A_3A_4)$
Or $(A_1A_2A_3)\ A_4$



A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

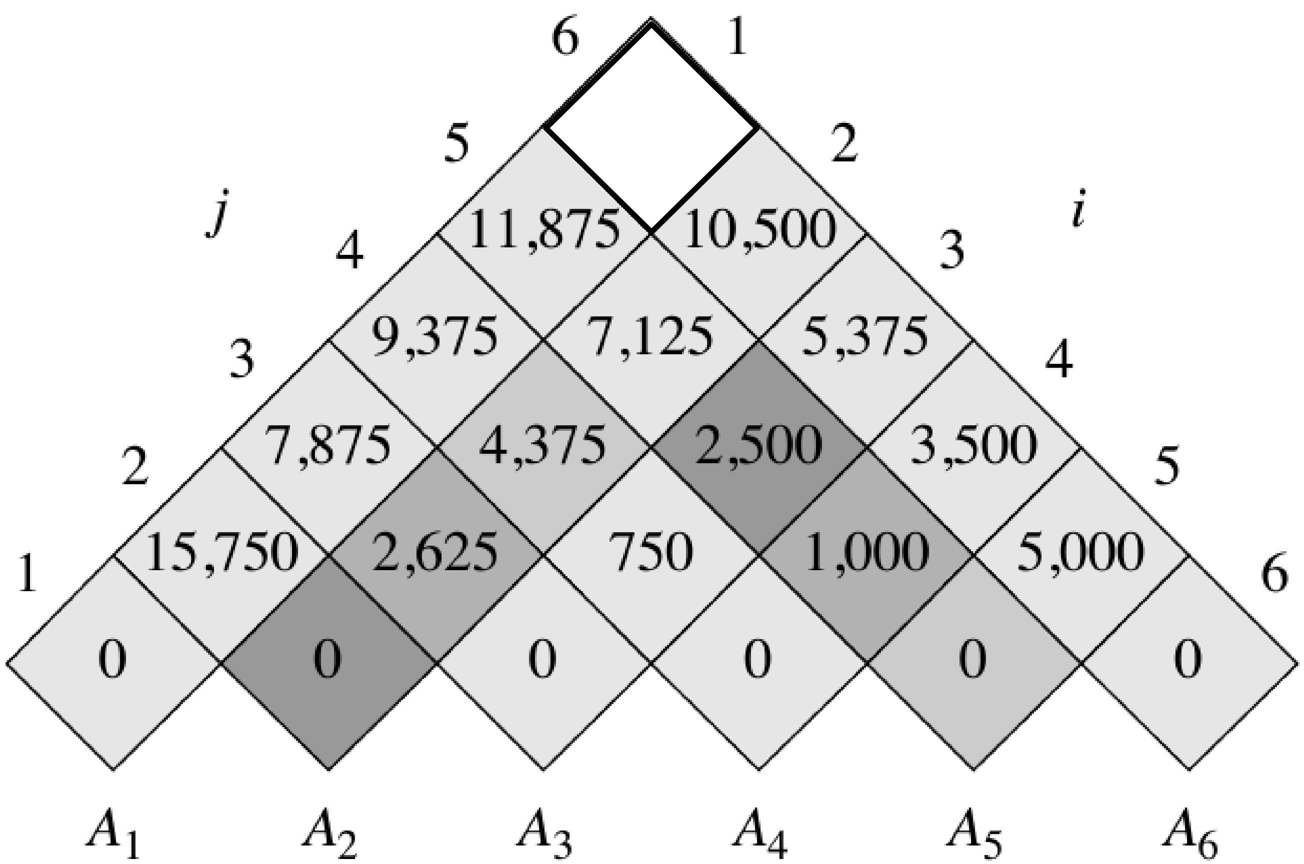
- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n



A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

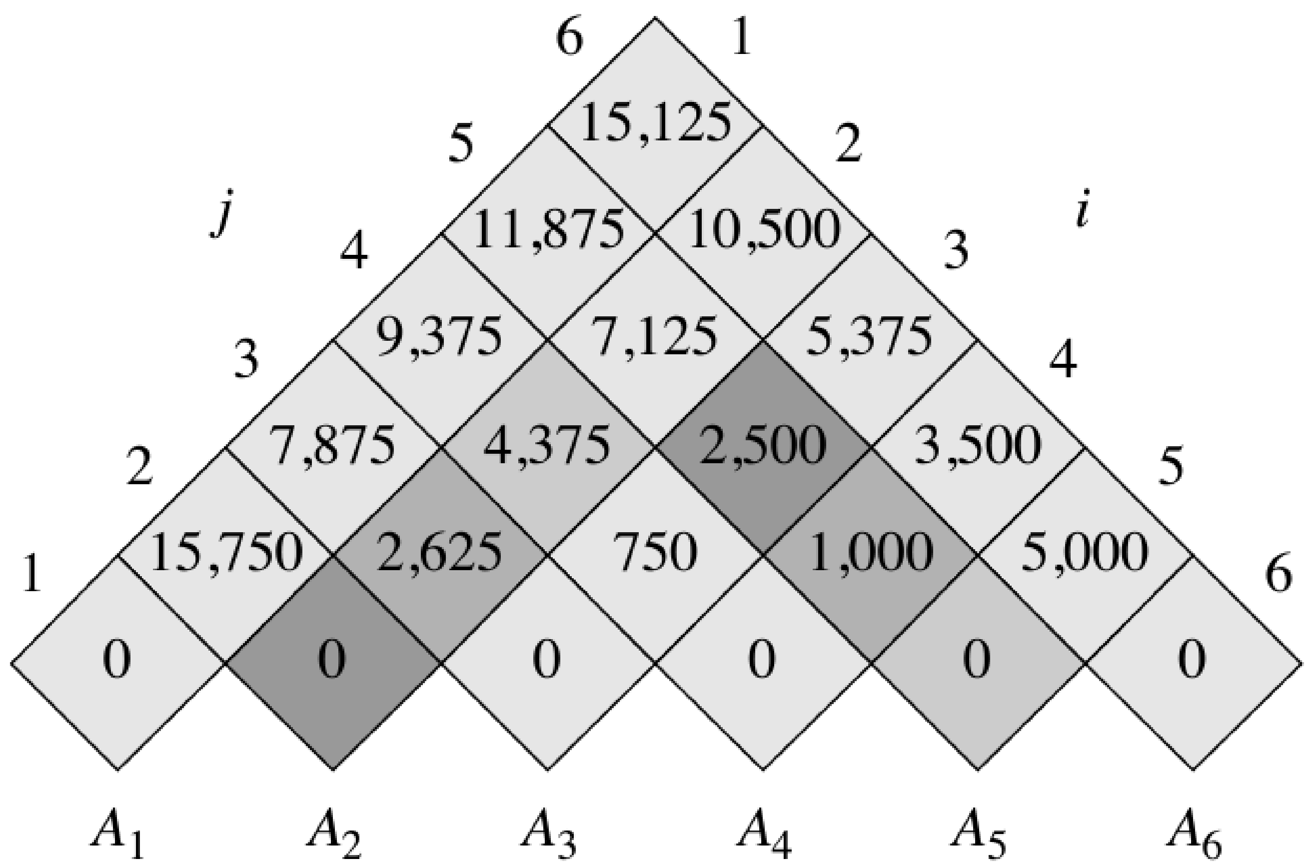
- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n



A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n



A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25

Matrix Chain Multiplication

MATRIX-CHAIN-ORDER(*p*)

```
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                  // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Longest Common Subsequence

$$S_1 = \mathrm{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$$
$$S_2 = \mathrm{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$$

- A string subsequence is an ordered set of characters (not necessarily consecutive)
- A common subsequence of two strings is a subsequence that exist in both strings.
- The longest common subsequence is the common subsequence of the maximum length.

Longest Common Subsequence

- Given two strings: $X = \langle x_1, x_2, \ldots, x_m \rangle$

  $$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

- Find the longest common subsequence of X and Y
  LCS(X,Y)

Longest Common Subsequence

- Given two strings:
$$X = \langle x_1, x_2, \ldots, x_m \rangle$$
$$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

- Find the longest common subsequence of X and Y
LCS(X,Y)
  - Brute force?

Longest Common Subsequence

- Given two strings:

$$X = \langle x_1, x_2, \ldots, x_m \rangle$$
$$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

- Find the longest common subsequence of X and Y LCS(X,Y)
  - Brute force? $O(n*2^m)$ or $O(m*2^n)$ [enumerate all subsequences of X and check in Y, or vice versa]

Longest Common Subsequence

- Given two strings: $$X = \langle x_1, x_2, \ldots, x_m \rangle$$ $$Y = \langle y_1, y_2, \ldots, y_n \rangle$$
- Find the longest common subsequence of X and Y LCS(X,Y)
  - Brute force? $O(n*2^m)$ or $O(m*2^n)$ [enumerate all subsequences of X and check in Y, or vice versa]
- Are smaller problems simpler?

Longest Common Subsequence

- Given two strings:

$$X = \langle x_1, x_2, \ldots, x_m \rangle$$
$$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

- Find the longest common subsequence of X and Y LCS(X,Y)
  - Brute force? $O(n*2^m)$ or $O(m*2^n)$ [enumerate all subsequences of X and check in Y, or vice versa]
- Are smaller problems simpler?
- Let's define string prefixes

$$X_i = \langle x_1, x_2, \ldots, x_i \rangle, \text{for } i = 0, 1, \ldots, m$$

- and same for Y_j for j = 0,1,…., n

Longest Common Subsequence

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

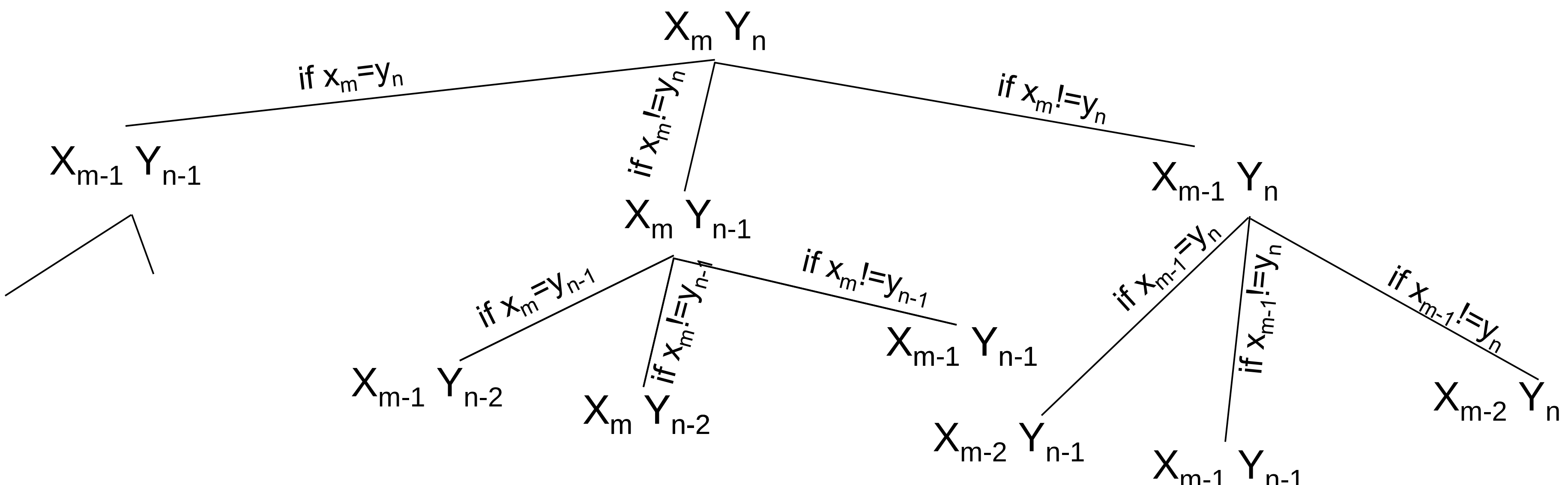
- Prove by contradiction

Longest Common Subsequence

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

- Prove by contradiction

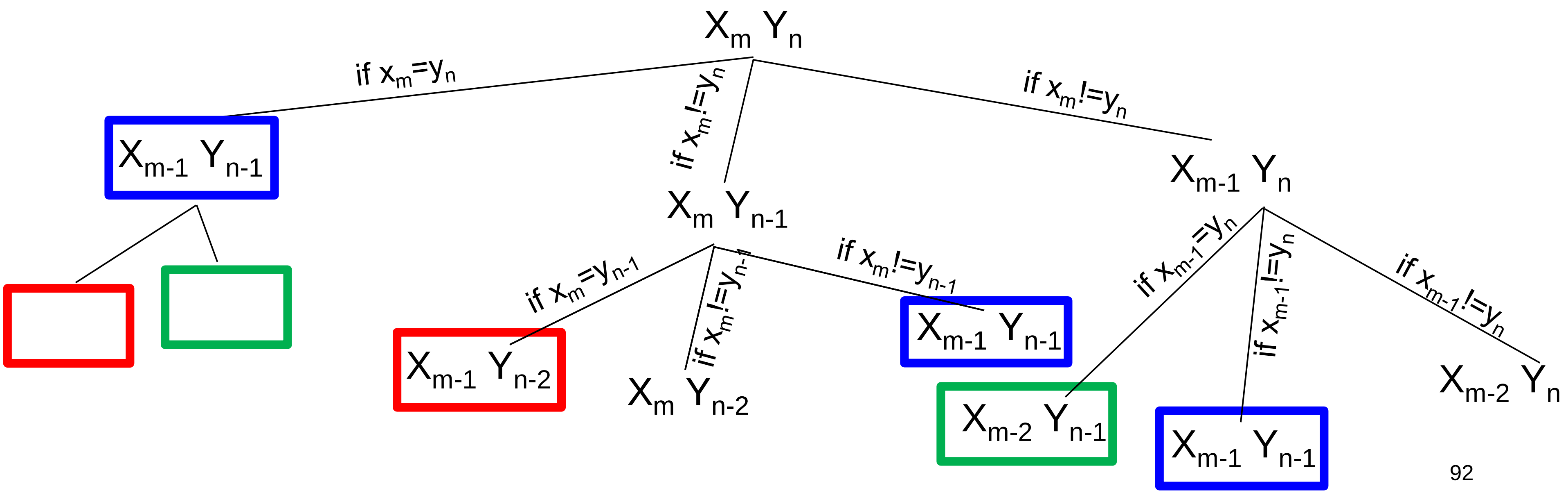


Longest Common Subsequence

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

- Prove by contradiction



Longest Common Subsequence

- Let c[i,j] is LCS length of X_i and Y_j

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \,. \end{cases}$$

Longest Common Subsequence

- Example: X=“CS141” Y=“CS111”

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

	“”	C	S	1	1	1
“”						
C						
S						
1						
4						
1						

Longest Common Subsequence

- Example: X="CS141"   Y="CS111"

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 , \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j , \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . \end{cases}$$

	""	C	S	1	1	1
""	0	0	0	0	0	0
C	0					
S	0					
1	0					
4	0					
1	0					

Longest Common Subsequence

- Example: X="CS141" Y="CS111"

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

	""	C	S	1	1	1
""	0	0	0	0	0	0
C	0	1	1	1	1	1
S	0	1	2	2	2	2
1	0	1	2	3	3	3
4	0	1	2	3	3	3
1	0	1	2	3	4	4

Longest Common Subsequence

```
LCS-LENGTH(X, Y)
m [redacted]ength
n [redacted]ength
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

Book Readings

- Ch. 15: 15.1-15.4