Time complexity of partition(A,p,r) – O(n)

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4       do if A[j] ≤ x
5            then i ← i + 1
6                 exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

Randomized quicksort

-A pivot element is selected at random (random sampling)

cf ranomization used in chapter 5 – random permutation

```
RANDOMIZED-PARTITION(A, p, r)
1  i ← RANDOM(p, r)
2  exchange A[r] ↔ A[i]
3  return PARTITION(A, p, r)
```

The new quicksort calls RANDOMIZED-PARTITION in place of PARTITION:

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2     then q ← RANDOMIZED-PARTITION(A, p, r)
3          RANDOMIZED-QUICKSORT(A, p, q − 1)
4          RANDOMIZED-QUICKSORT(A, q + 1, r)
```

The running time of QUICKSORT is dominated by the time spent in the PARTITION procedure. Each time the PARTITION procedure is called, a pivot element is selected, and this element is never included in any future recursive calls to QUICKSORT and PARTITION. Thus, there can be at most n calls to PARTITION over the entire execution of the quicksort algorithm. One call to PARTITION takes $O(1)$ time plus an amount of time that is proportional to the number of iterations of the **for** loop in lines 3–6. Each iteration of this **for** loop performs a comparison in line 4, comparing the pivot element to another element of the array A. Therefore, if we can count the total number of times that line 4 is executed, we can bound the total time spent in the **for** loop during the entire execution of QUICKSORT.

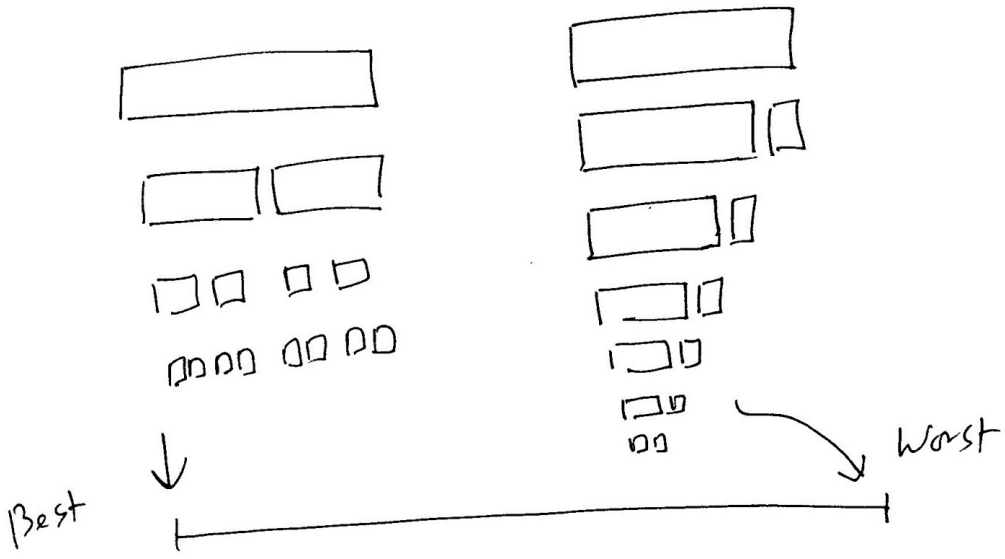
Lemma 7.1

Let X be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array. Then the running time of QUICKSORT is $O(n + X)$.

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4       do if A[j] ≤ x
5            then i ← i + 1
6                 exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

X a random variable

the total number of comparisons performed in all calls to partition



a spectrum of possibilities

Assume that all numbers are distinct.

Let's call ith smallest number z_i.

Ex. $\{4, 5, 1, 8, 7, 6\}$

$z_2 = 4, \quad z_6 = 8$

Let's define $Z_{ij} = \{z_i, z_{i+1}, \cdots, z_j\}$

Ex $\rightarrow Z_{25} = \{4, 5, 6, 7\}$

When does the algorithm compare z_i, z_j

2 possibilies

either z_i is a pivot, or

z_j is a pivot

When does the algorithm compare z_i, z_j

2 possibilies

either z_i is a pivot, or

z_j is a pivot

an indicator random variable X_{ij}

def. $X_{ij} = I\{z_i \text{ is compared to } z_j\}$

the total numbe of comparisons X

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

$z_{i,j} = \{ \quad \}$

4, 5, 1, 8, 7, 6

$z_{2,4} = \{4, 5, 6\}$

↙

2nd smallest

→ 4th smallest

z_i, z_j $\qquad$ Exp(X)

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

indicate rand variable

$X_{i,j}$ { 1 if z_i, z_j are compared; 0 otherwise.

$$Exp\left(\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \right)$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Exp(X_{i,j})$$

z_i, z_j

n-1 n

$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[z_i, z_j \text{ are compared}]$

z_i pivot or z_j pivot

$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$

$\ll O(\log n)$

$z_{i,j}$ $\quad j-i+1$

$= \sum_{k=1}^{n} \frac{2}{k+1}$

$\hookrightarrow O(n \log n)$

$$E(X) = E\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right)$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(X_{ij})$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr\{z_i \text{ is compared to } z_j\}$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \left(\frac{1}{j-i+1} + \frac{1}{j-i+1}\right)$$

$$= 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{1}{j-i+1}$$

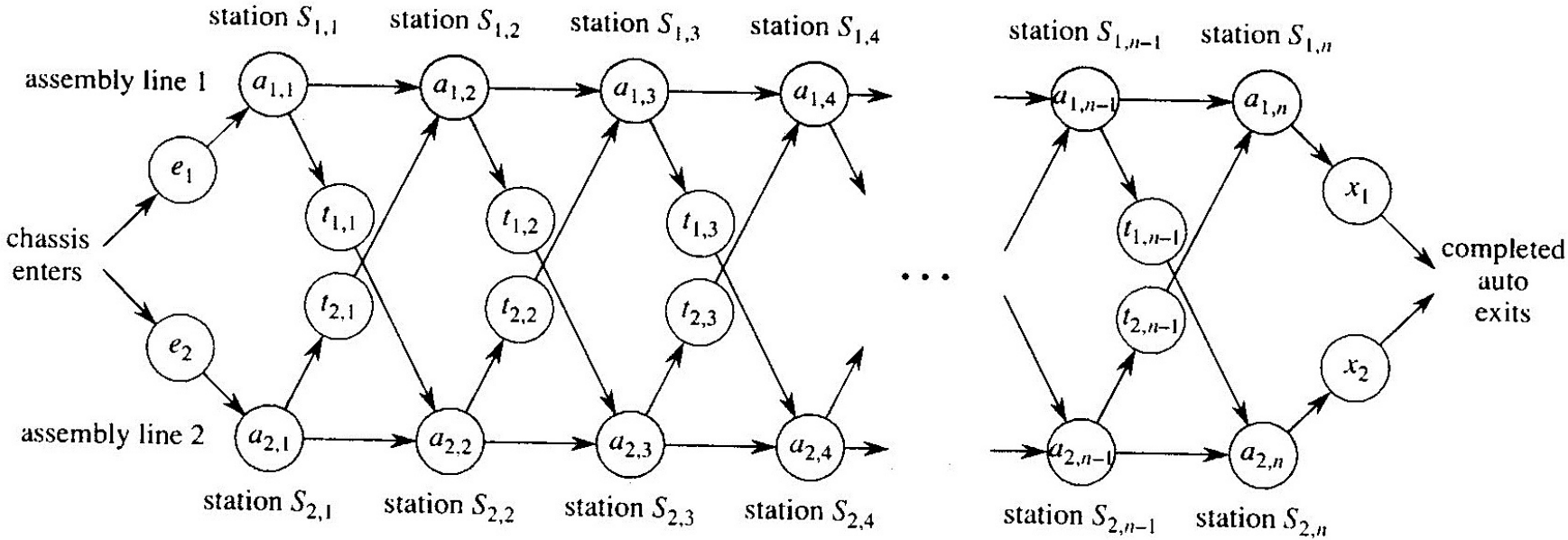
$$= 2\sum_{i=1}^{n-1}\sum_{k=1}^{n-i} \frac{1}{k+1} < 2\sum_{i=1}^{n-1}\left(\sum_{k=1}^{n} \frac{1}{k}\right)$$

$O(\lg n)$

$$= 2\sum_{i=1}^{n-1} O(\lg n)$$

$$= O(n \lg n)$$

Assembly line scheduling problem – an optimization problem

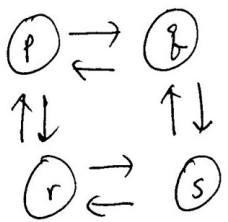


Structural properties of the assembly line scheduling problem

1. Optimal substructure property – an optimal solution consists of optimal solutions of subproblems

2. Overlapping subproblems property – the same subproblem may be repeatedly solved again recursively

optimal substructure property



Shortest path from p to s

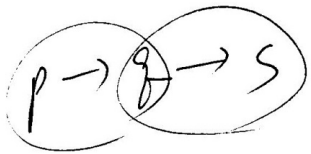
problem X

an optimal sol

$p \to q \to s$

$p \to r \to s$

the structure of an optimal sol



consists of optimal sols of subproblems of X

- subproblems of X

① an optimal path from p to q

② an optimal path from q to s

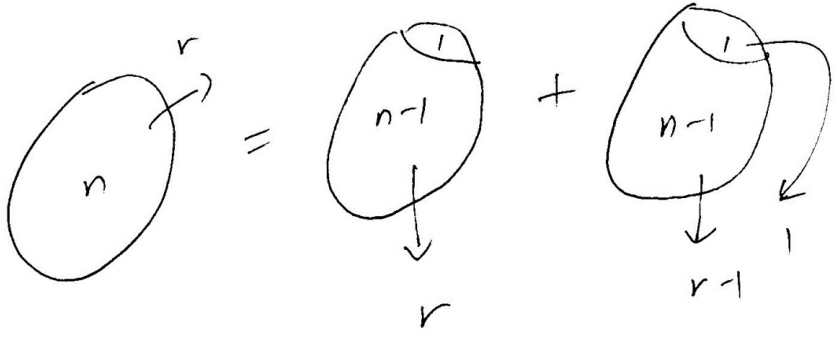
an optimal sol for ① : $p \to q$

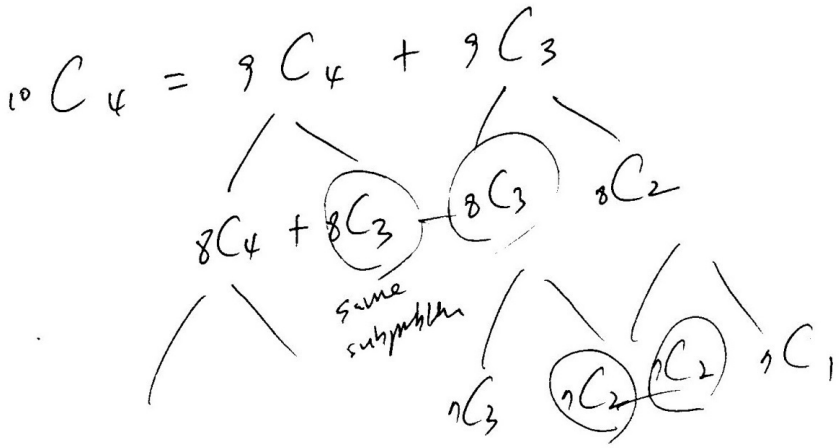
an optimal sol for ② : $q \to s$

overlapping subproblems

problem: ${}_nC_r = {}_{n-1}C_r + {}_{n-1}C_{r-1}$

(subproblem) (subproblem)





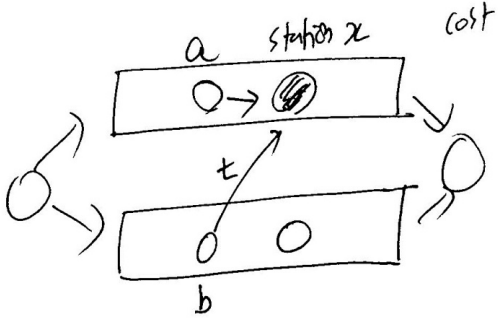
the same subproblem is repeated

solved recursively

again and again

optimal substructure property

of the assembly line scheduling problem



claim (optimal substructure property):

an optimal solution towards x.

i.e. a cheapest ~~way~~ cost towards x

$= \min(A, B) + y$

A: a cheapest cost towards a

B: a cheapest cost towards b + transfer cost (t)

Proof: What if NOT?

Proof: what if NOT?

That is,

an optimal solution towards x is $\min(A, B) + y$, But

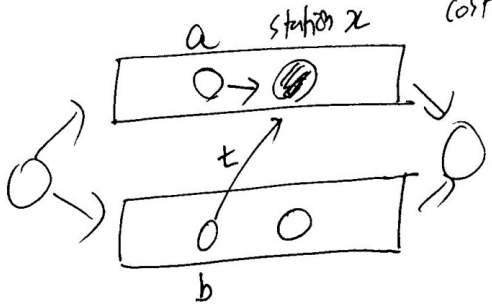
case $A < B$ (A is not an optimal solution towards a, OR

case $B < A$ B is not an optimal solution towards b

in case $A = B$, $A (= B)$ is not an optimal solution toward b.

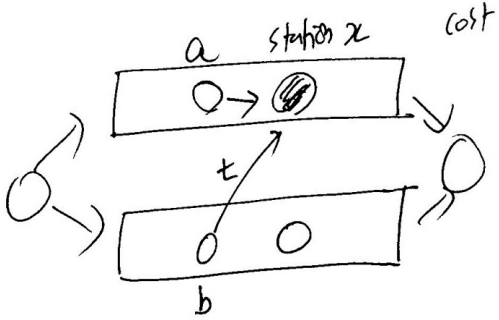
Contradiction

optimal substructure property of the assembly line scheduling problem



Contradiction

optimal substructure property of the assembly line scheduling problem



case $A < B$

there exists a better solution towards x than

$\min(A, B) + y = A + y$

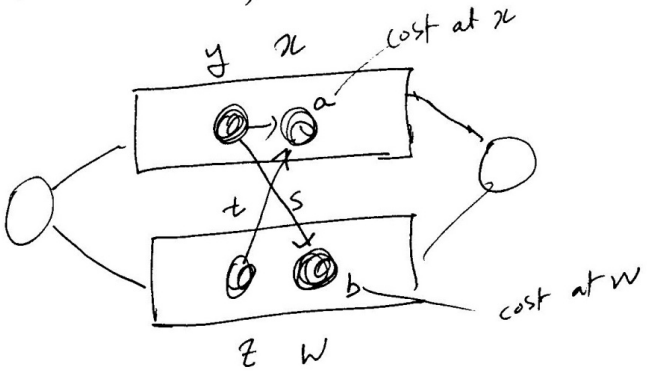
But, we assumed that we were given an optimal solution towards x initially.

case $B < A$, $A = B$

Similar.

overlapping subproblems property

of the assembly line scheduling problem



an optimal solution towards x

$= \min(A, B) + a$

an optimal solution towards w

$= \min(C, D) + b$

the same subproblem needs to be computed again and again

A: an optimal solution towards y

B: an optimal solution towards z + transfer cost (t)

(=

C: an optimal solution towards z

D: an optimal solution towards y + transfer cost (s)

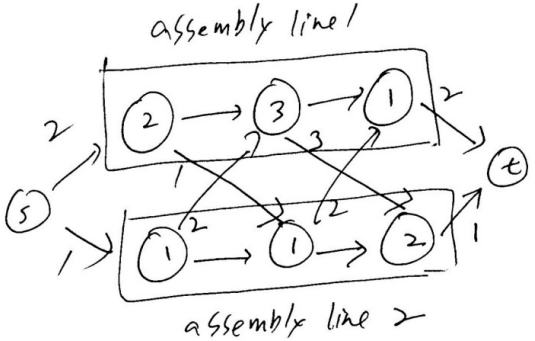
How can we “exploit” these structural properties of the assembly line scheduling problem?

Optimal substucture property ⍰ an optimal solution can be found by solving smaller problems and combining those solutions for the smaller problems

(because an optimal solution MUST consist of optimal solutions of its subproblems)

Overlapping subproblems property ⍰ we can store optimal solutions in a table so that we don’t have to solve the same subproblems again and again repeatedly

an instance of the assembly line scheduling problem



optimal solutions towards each stations

4	7	6
2	3	5

assembly line 1

assembly line 2

2+2 1+1

Time complexity
O(n)

$\min(4+3, 2+2+3) = 7$

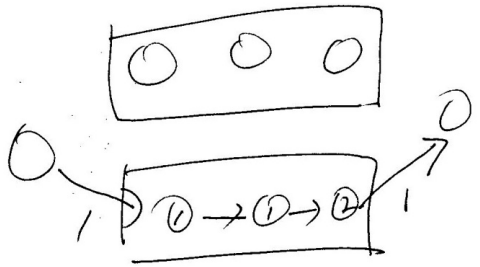
$\min(2+1, 4+1+1) = 3$

$\min(7+1, 3+2+1) = 6$

$\min(3+2, 7+3+2) = 5$

an optimal solution towards t

$= \min(6^{+2}, 5^{+1}) = 6$



10/11

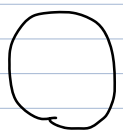
최적의 알고리즘 설계과 분석을 위한 대표적 방법들

ch 15 Dynamic Programming (DP)

ch 16 Greedy Algorithms (GA)

Similar

1. Incremental App
2. Divide & conquer
3. using DS



good example.

Good ex

Similar

Heap sort

top down approach → compute problem

Bottom up approach //

Optimization problem

↓

① maximizing some value → LCS.

② minimizing some value.

DP → 전역적 선택이 필요한 최적화 문제에 적용.

GA → 지역적(locally) 선택을 하는 것.

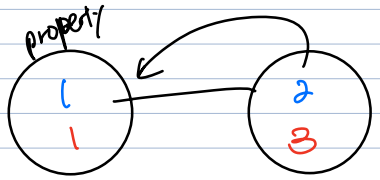
★ if GA is applicable then DP //

★ if both are applicable GA has a better solution.

DP
- ① optimal substructure property
- ② overlapping subproblems property

Th16.1
- ③ greedy choice property

GA: structural property



두 property가 모두 있어야지 DP

DP or GA working.

16.4 → a matroid ← GA

(0,0)

if problem is a matroid structure the GA ALWAYS work.

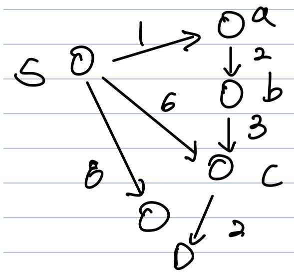
① 최적해 구조 특성

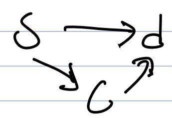
Def

An Optimal Sub Structure property → 작게 쪼개서

- an optimal sol of π

~~the~~ "하나만 있는게 아니다."



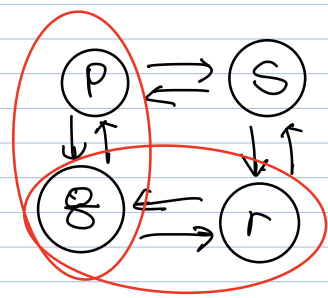


s에서 d로 가는 수 있는 여러가지 길

s → d

s → c → d.

Ex



a shortest path from p to r

p → s → r

p → q → r

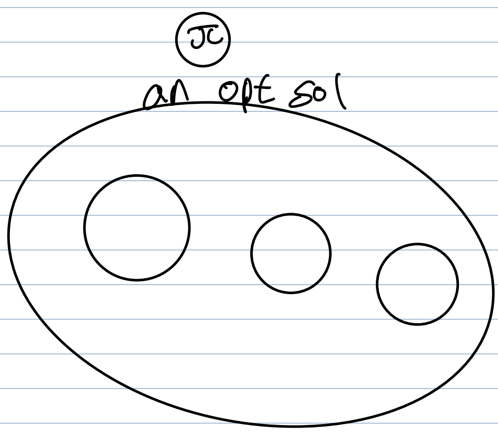
p → q
q → r

an opt sol

p to q 까지의 opt sol

q to r 까지의 opt sol

consists of optimal solution of π's subproblems



(2)

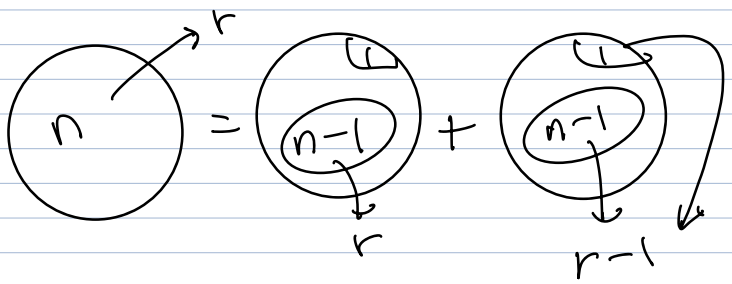
Overlapping subproblem property

$${}_9C_4 = {}_8C_4 + {}_8C_3$$

$${}_8C_4 + {}_8C_3 \qquad {}_7C_3 + {}_7C_2$$

Example #1

$${}_nC_r = {}_{n-1}C_r + {}_{n-1}C_{r-1}$$



Example #2 - 피보나치 수열

$$f(n) = f(n-1) + f(n-2)$$

$$f(0) = f(1) = 1$$

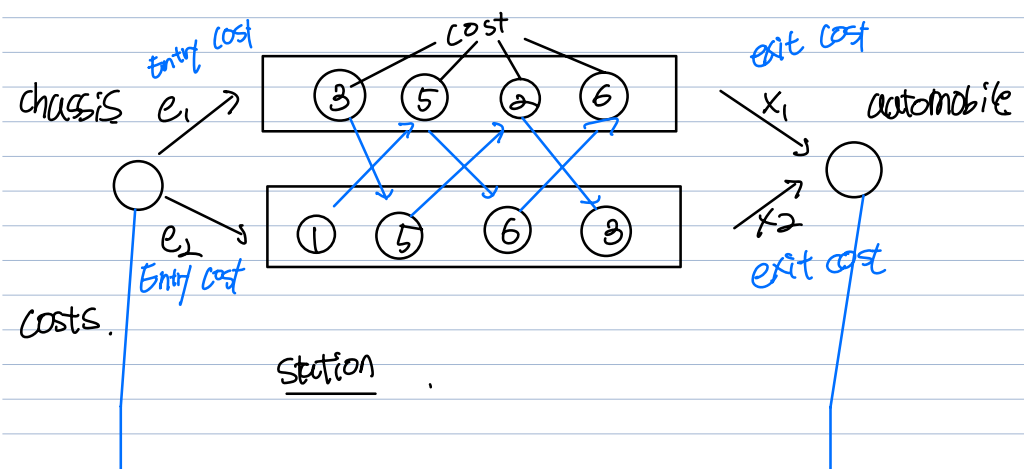
$$f(10) = f(9) + f(8)$$

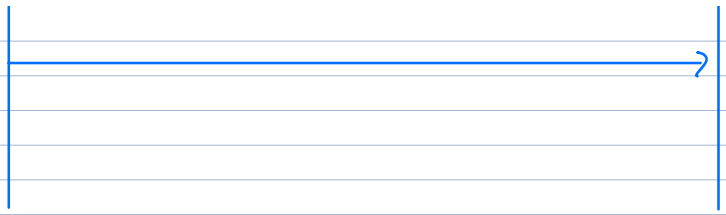
$$\downarrow$$

$$f(8) + f(7)$$

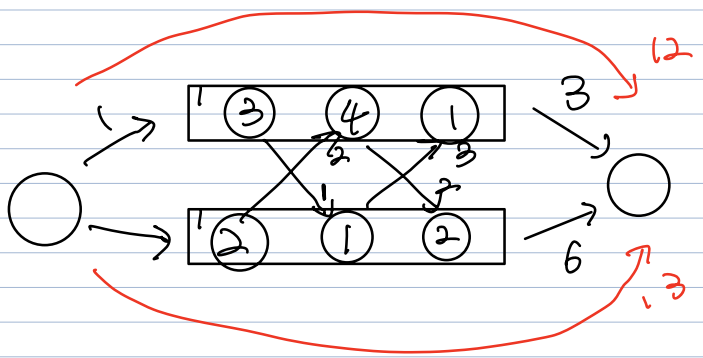
- Example problem

Assembly line scheduling problem





Cheapest cost
fastest



brute force.
↳ requires $O(2^n)$

n stations
2^n (8)

$n = 100$
2^{100} impracticable

이렇게 사용할 수 없다

with X space

- opt S
- over S_1

} DP

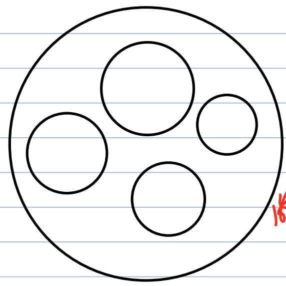
$O(n)$ ↓
space ↑

이렇게 사용할 수 있다면 DP need table,
so. space 필요해야한다.

↓Time vs ↑space
Trade off

opt substructure property

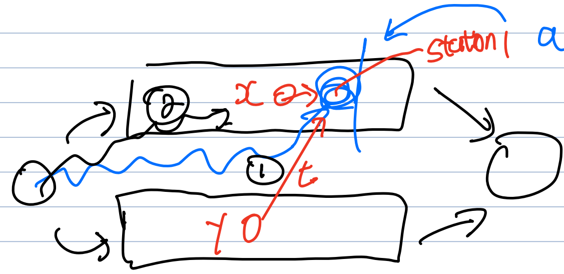
an opt sol → 어떻게 찾았는지 [illegible]
이것은 주어졌다고 생각



주어진 opt sol property를
이 집합에 존재한다고 생각

& "shape" of an opt sol — optimal sol is given.

station 1 — a fastest way



an opt sol for S_{15} — 5th station

O_{15}

first: ①

① $x < y+t$

Claim: $O_{15} = \min(x,\ y+t) + \text{cost } S_{15}$

② $x \geq y+t$

① $x + \text{cost } S_{15}$

② $y+t + \text{cost } S_{15}$

ⓧ 1

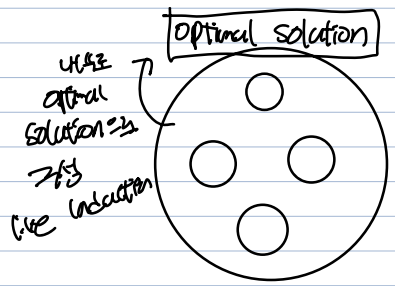
what if not

10/17

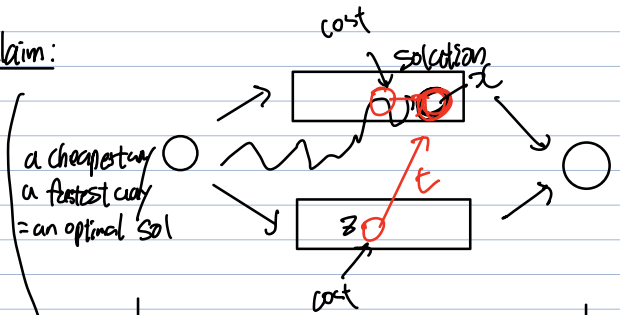
CH15

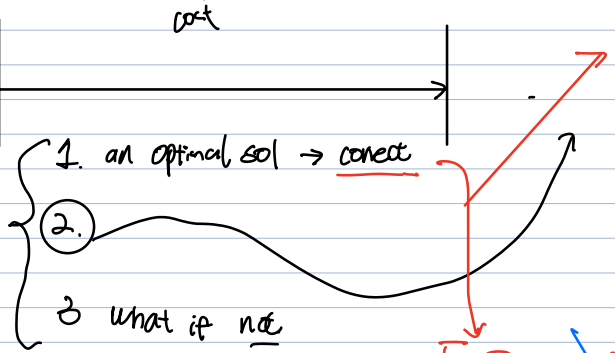
Optimal substructure property → the shape of an optimal solution

Assembly line scheduling problem



Claim:





opt sol = min (y , z+t) + x

y + x must be an opt sol

① y < z+t

② y > z+t

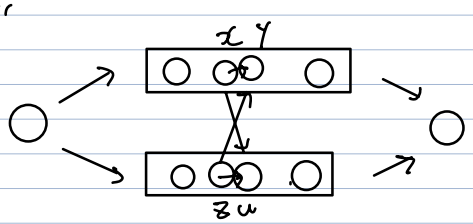
③ y = z+t

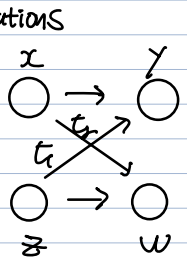
[① an opt sol ② ①이면 ③이 성립하지 않음
③ y is not an opt sol] → contradiction

Optimal substructure property
Overlapping subproblems property
) Dynamic programming bottom up.

Overlapping subproblems property ← some observation

Claim,





- for each station
find an opt, solution to solve this problem

An optimal solution towards y

① opt sol - x } → a
② opt sol - z } → b

same

A optimal solution towards w

① opt sol - z : c
② opt sol - x : d

$\min(a, b+t_i)$

$\min(c, d+t_2)$

같으므로

table을 이용하여

값을 저장

→ 다음계산 때 저장한 값을 꺼내서 사용함으로써

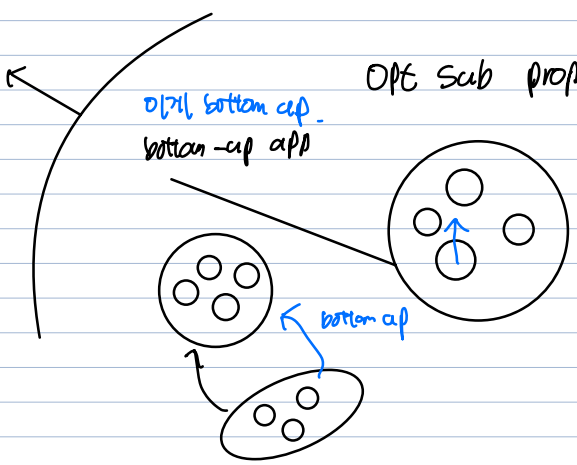
이게 → "Dynamic programming"

ALSP

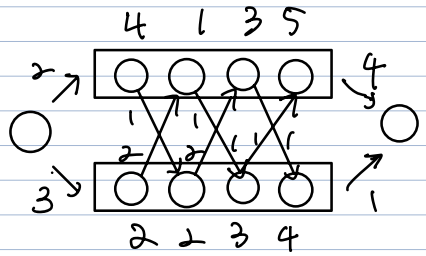
Opt sub property

이게 bottom up

bottom-up app



Example.



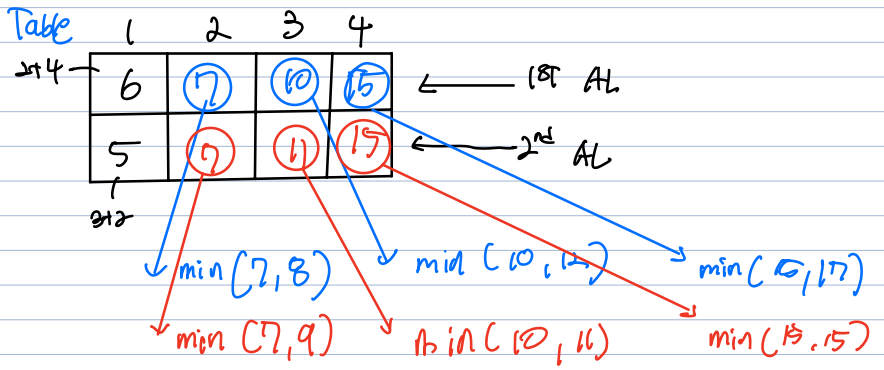
$2^4 = 16$

a brute way $O(2^n)$

↓

Dp를 사용하면 $O(n)$ 2×4

but time - space trade off 존재



Example #2

Matrix, multiplication

A , B

p×q q×r

save

① first 3rd

$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

2×3 × 2×1

can't multiply

---

② $A \cdot B \neq B \cdot A$

③ A : p×q, B : q×r

$A \cdot B = p \times r$

# of scalar multiplication

$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$

2×3 3×1

$p \times q \times r$

2×3×1 → 6

2 · 1

$[(1\cdot1+1\cdot2+1\cdot1)\ \ (1\cdot1+2\cdot2+1\cdot1)]$

④ Associative $(AB)C = A(BC)$

$(AB)C = A(BC)$

Ex) A: 2×5
B: 5×3
C: 3×10
a dimension

# of scalar multiplications

$(AB)C$ | $AB = \boxed{2\times5\times3} = 30$

2×3 · C, 3×10

$2\times3\times10 = 60$

$30+60 = \boxed{90}$

$A(BC)$ | $(BC) = \boxed{5\times3\times10} = 150$ → 이미 90보다 크다.

5×10

$2\times5\times10 = 100$

$100+250 = \boxed{350}$

결론 강의
계산 순서에 따라 계산횟수가 다르기 때문에 순서가 매우 중요!

Matrix chain multiplication pub $n \geq 1$

input: a list of compatible n matrices

output: ① an optimal way of multiplying (parenthesis) all matrices
② an optimal # of scalar mults.

정은샘

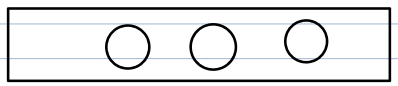
Def a chain of metrics is fully parenthesized

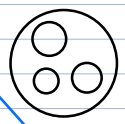
S ① any 1 matrix

Ex) $((A)(B))(C))$

20② (b, o)
과제

1. optimal substructure property

1. an optimal solution is given (correct)
2. 
3. what it NOT?



$A_1, A_2 \cdots A_n$

((x)(y)) the shape of opt

괄호그대로 나뉜다.

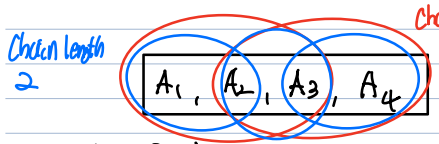
x: opt
y: opt

x y optimal solution contains optimal solution.

Contradiction
① x is Not opt
② y is Not opt
③ x, y are Not opt

어떤 경우 나온 solution이 있더라도 그러면 1. 전제 "an optimal solution is given"이 틀린다.

2. Overlapping Subproblems property → 값들이 겹치는게 있다.



→ Chain length 4

$A_1 = 2 \times 3$
$A_2 = 3 \times 4$
$A_3 = 4 \times 5$
$A_4 = 5 \times 6$

1. – Chain length 4
2 – Chain length 3
3 – Chain length 2

min # scalar multiplication

CL = 2

$A_1 \times A_2$: $2 \times 3 \times 4$

$A_2 \times A_3$: $3 \times 4 \times 5$

$A_3 \times A_4$: $4 \times 5 \times 6$

CL = 3

min

$A_1(A_2A_3)$

$(A_1A_2)A_3$

min

$A_1A_2A_3$

$A_1A_2A_3A_4$

값들이 같다!

$(A_2A_3)A_4$

$(A_1A_2A_3)A_4$

$A_2A_3A_4$

$A_2(A_3A_4)$

이것도

② $(A_1A_2)(A_3A_4)$

③ $A_1(A_2A_3A_4)$

table 값들이 결과적 table을 만들어 저장하면

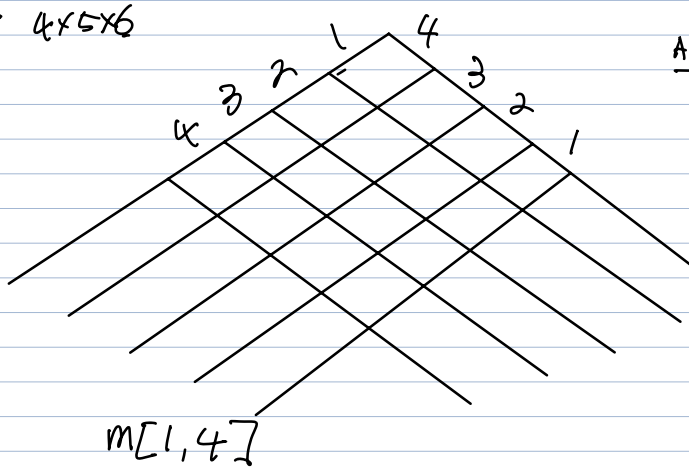
O.S.P 계산 시간이 준다!

Dynamic programming.

$A_1 \times A_2 = 2 \times 3 \times 4$

$A_2 \times A_3 = 3 \times 4 \times 5$

$A_3 \times A_4 = 4 \times 5 \times 6$



$A_1 A_2 A_3 A_4$

$m[1,4]$ # of Optimal scalar multiplications

$m[1,2] = 24$

$m[2,3] = 60$

$m[3,4] = 120$

10/18 화

$A_1 = 2 \times 3$

$A_2 = 3 \times 4$

$A_3 = 4 \times 3$

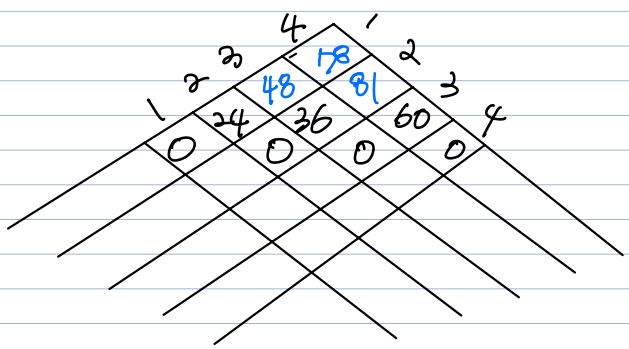
$A_4 = 3 \times 5$

$m[1,1] = 0$ $m[1,2] = 24$

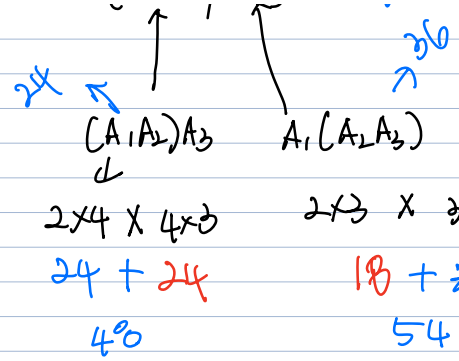
$m[2,2] = 0$ $m[2,3] = 36$

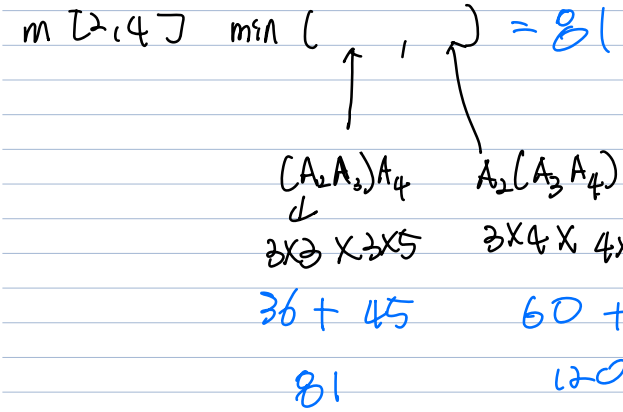
$m[3,3] = 0$ $m[3,4] = 60$

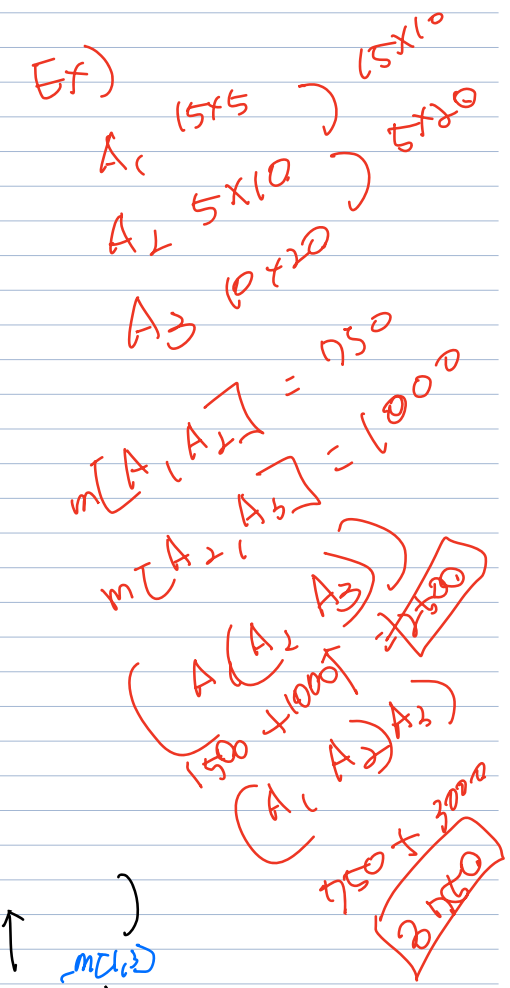
$m[4,4] = 0$

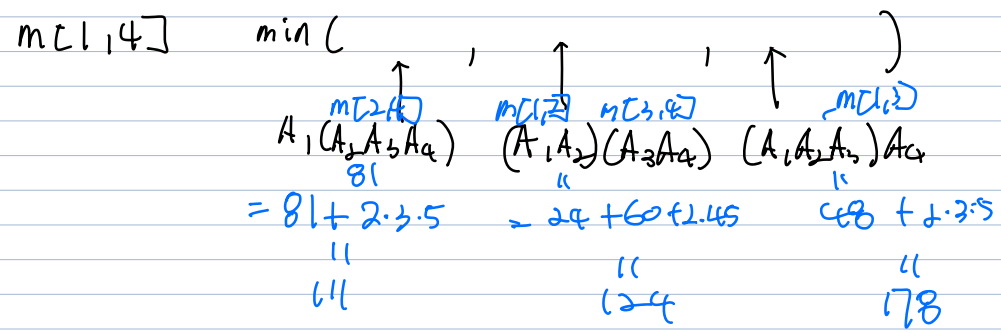


$m[1,3]$ $\min(\quad,\quad) = 48$


24
36
(A1A2)A3
A1(A2A3)
2x4 x 4x3
2x3 x 3x3
24 + 24
18 + 36
48
54


m[2,4] min(,) = 81
(A2A3)A4
A2(A3A4)
2x3 x 3x5
3x4 x 4x5
36 + 45
60 + 60
81
120


Ex)
A1 15x5
A2 5x10
A3 10x20
15x10
5x20
m[A1,A2] = 750
m[A2,A3] = 1000
A(A2A3)
1500 + 1000 = 2500
(A1A2)A3
750 + 3000
3750


m[1,4] min(, ,)
m[2,4]
m[1,2]
m[3,4]
m[1,3]
A1(A2A3A4)
(A1A2)(A3A4)
(A1A2A3)A4
81
24
48
= 81 + 2·3·5
= 24 + 60 + 2·45
= 48 + 2·3·5
111
124
178

This approach is called dynamic programming

-Programming means “tabular method”: we use a table

It turns out that

if an optimization problem possesses both

(1) Optimal substructure property, and

(2) Overlapping subproblems property

then

we “can” use dynamic programming to solve the problem

Matrix chain multiplication – a second example problem that can be solvable using dynamic programming

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

$2 \times 3 \qquad 3 \times 1$

dimension

# of rows: 2

# of columns: 3

$$\begin{pmatrix} 2\times1 + 1\times1 + 3\times2 \\ 1\times1 + 1\times1 \times 2\times2 \end{pmatrix}$$

2×1

In general $p \times q \quad q \times r \quad p \times r$

$$(\ )\times(\ ) = (\ )$$

# of scalar multiplications $= p \times q \times r$

matrix multiplication is associative

$$(AB)C = A(BC)$$

But, the # of scalar multiplications may be different.

Ex

A: 2×3
B: 3×4
C: 4×2

dimensions

AB : $2\times3\times4 = 24$

2×4 matrix

(AB)C : $24 + 2\times4\times2 = 40$

BC : $3\times4\times2 = 24$

3×2 matrix

A(BC) : $2\times3\times2 + 24 = 36$

the # of scalar multiplications

(AB)C	A(BC)
40	36

better

Matrix chain multiplication problem

Input: of matrix-chain multiplication. We are given a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices to be multiplied, and we wish to compute the product

$$A_1 A_2 \cdots A_n .$$

ambiguities in how the matrices are multiplied together. A product of matrices is ***fully parenthesized*** if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses. Matrix multiplication is associative, and so all parenthesizations yield the same product. For example, if the chain of matrices is $\langle A_1, A_2, A_3, A_4 \rangle$, the product $A_1 A_2 A_3 A_4$ can be fully parenthesized in five distinct ways:

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

Output: the total number of possible “full parenthesizations”

an instance of the matrix chain multiplication problem

A_1, A_2, A_3 : 2

① $((A_1 A_2)(A_3))$

② $((A_1)(A_2 A_3))$

optimal substructure property

$A_1, A_2, \cdots, A_n$

Assume that an optimal solution; that is an optimal way (= a way with min # of scalar multiplications) of multiplying $A_1, A_2, \cdots, A_n$.

$A_1, A_2, \cdots, A_n$

Assume that an optimal solution; that is an optimal way (= a way with min # of scalar multiplications) of multiplying $A_1, A_2, \cdots, A_n$.

Right before one final parenthesization, i.e. $(A_1 A_2 \cdots A_n)$ there must exist 2 full parenthesizations; i.e.

$((A_1 \cdots)(\cdots A_n))$

Claim $((A_1 \cdots) \uparrow (\cdots A_n))$

we do not know exactly where we have this split,

But, it must be that

Each part should be multiplied in an optimal way.

that is, multiplied with min # of scalar multiplications.

What if NOT?

→ Contradiction

What if NOT ?

That is,

we have an optimal solution for $(A_1, \cdots, A_n)$ $((\underbrace{(A_1 \cdots}_{x})(\underbrace{\cdots A_n}_{y}))$

But,

NOT (each subpart is multiplied in an optimal way)

$((\underbrace{(A_1 \cdots}_{x})(\underbrace{\cdots A_n}_{y}))$

NOT (each subpart is multiplied in an optimal way)

$\equiv$

① x is multiplied in an optimal way, But y is NOT

② y is multiplied in an optimal way, But x is NOT

③ NONE of x, y are multiplied in opt ways.

NOT (each subpart is multiplied in an optimal way)

$((A_1 \cdots)(\cdots A_n))$

x $\quad$ y

$\equiv$ ① x is multiplied in an optimal way, But y is not

② y is multiplied in an optimal way, But x is N.T

③ N.N.E. of x, y are multiplied in opt ways.

② ③ Similar

Therefore, an optimal for $(A_1 \cdots A_n)$ consists of optimal sols of its subproblems.

Overlapping subproblems property

Ex // an optimal way of multiplying $A_1 A_2 A_3 A_4$

$A_1 : 2 \times 3$
$A_2 : 3 \times 4$
$A_3 : 4 \times 3$
$A_4 : 3 \times 5$

$A_1 A_2 : 2 \times 3 \times 4$
$A_2 A_3 : 3 \times 4 \times 3$
$A_3 A_4 : 4 \times 3 \times 5$

min # of scalar multiplications

$A_1 A_2 A_3$ — 2 possibilities
$A_2 A_3 A_4$ — "

We need to know # of scalar multiplications

$(A_1 A_2) A_3$, $A_1 (A_2 A_3)$

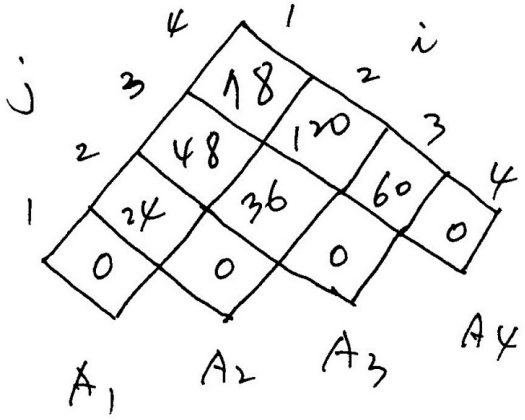
$(A_2 A_3) A_4$, $A_2 (A_3 A_4)$

an optimal sol for each needs to solve the same subproblem : i.e. min # of scalar multiplications for $A_2 A_3$

A_1: 2×3 A_2: 3×4

A_3: 4×3 A_4: 3×5

m



bottom-up fashion

m[i,j]: min # of scalar multiplications needed to compute the matrix

$A_i \cdots j$

m[1,2] : 2×3×4 = 24

m[2,3] : 3×4×3 = 36

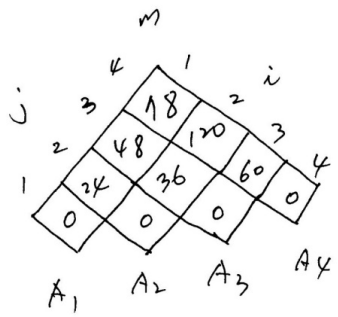
m[3,4] : 4×3×5 = 60

m[1,3]

m[2,4]

A_1: 2×3 A_2: 3×4

A_3: 4×3 A_4: 3×5



bottom-up fashion

$A_2A_3A_4$ m[2,4]

min

$(A_2A_3)A_4$ min[2,3] + 3×3×5 = 36 + 45 = 81

3×3 3×5

$A_2(A_3A_4)$ 3×4×5 + min[3,4] = 60 + 60 = 120

3×4 4×5

$\therefore$ m[2,4] = 120 **[not correct]**

$A_1A_2A_3$ m[1,3]

min

$(A_1A_2)A_3$ m[1,2] + 2×4×3 = 24 + 24 = 48

2×4 4×3

$A_1(A_2A_3)$ 2×3×3 + m[2,3] = 18 + 36 = 54

2×3 3×3

$\therefore$ m[1,3] = 48

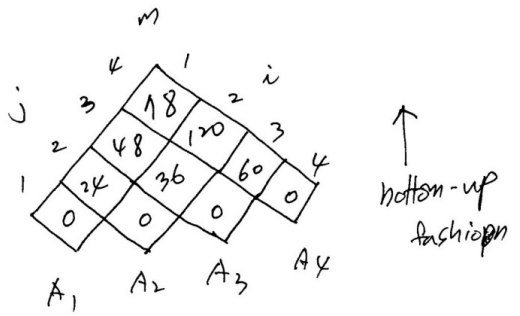
Correction – m[2,4] = 81

$m[1,4]$ $A_1A_2A_3A_4$

A_1: 2×3 A_2: 3×4

A_3: 4×3 A_4: 3×5

$$\min \begin{cases} m[1,1] + m[2,4] + 2\times3\times5 \\ m[1,2] + m[3,4] + 2\times4\times5 \\ m[1,3] + m[4,4] + 2\times3\times5 \end{cases}$$



$A_1A_2A_3A_4$

$(A_1)(A_2A_3A_4)$

$(A_1A_2)(A_3A_4)$

$(A_1A_2A_3)(A_4)$

$0 + 120 + 30 = 150$

$24 + 60 + 40 = 124$

$48 + 0 + 30 = 18$

Fig 15.3 - a similar example

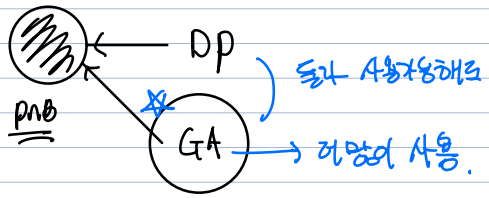
Homework #2 – due Mar 28, Monday midnight

Email submission (PDF format only – handwriting / typing – both are okay)

(1) Construct the table m for multiplying the chain of matrices: A1, A2, A3, A4, A5 in which the dimensions of the matrices are as follows:

A1: 2 by 4
A2: 4 by 5
A3: 5 by 6
A4: 6 by 2
A5: 2 by 3

(2) Assuming that there are n matrices, analyze the time complexity of this construction.



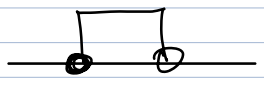
둘다 사용자정해도

라이브러리 사용.

Ch 16.1

Activity selection problem

$[s_i, f_i)$



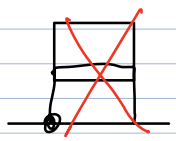
input : $a_1, a_2, \cdots a_n$

output: ① max # of compatible schedule (not overlap)

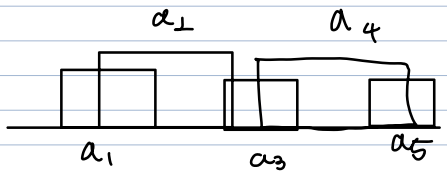
② a set of max # of

1. classroom (one exclusive resource)

cannot be shared.



Ex)



max # compatible. 3

$\{a_1, a_3, a_5\}$

10/31

ch 16 GA

Activity selection problem

16.5 . unit test scheduling problem

ch 23. prim's Alg
Kruskal's Alg

ch 25 Dijkstra's Alg

input : n activities

1. resources exclusively

included

- classes S, excluded.
- 1 class room

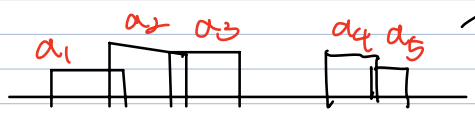
output : max # of compatible activities. Assumption : $f_1 \le f_2 \le \cdots \le f_n$

CLA 16.1

① optimal substructure property

② Greedy choice property

Ex



① 3

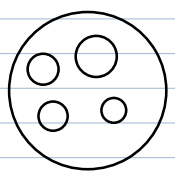
② {a1, a3, a4 }
{a1, a3, a5 }

① optimal substructure property

② Greedy choice property

③ what if not

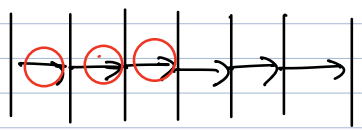
optimal solution



Greedy choice property

select → what looks best at that moment

not global

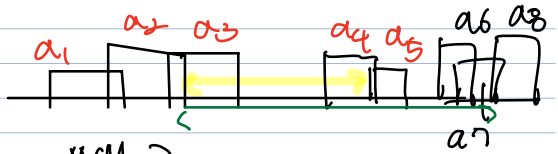


{ ① 0/1
② fractional

↓

greedy problem

Optimal substructure property.



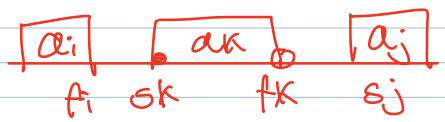
- MCM
- ALSP
- LCS

} a subproblem defines relatively easy.

a notation

$$S_{ij} = \{ a_k \mid f_i \le S_k < f_k \le S_j \}$$

$S_k \neq f_k$



$S_{25} = \{ \phi \}$

$S_{28} = \{ a_4, a_5, a_6 \}$ ↓ a subproblem

$S_{0\infty} = \{ \}$ entire problem.

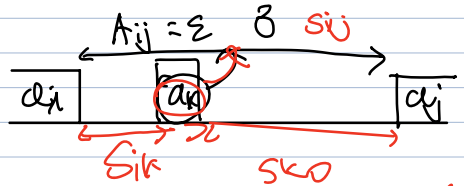
Optimal substructure property

1. an optimal sol for $S_{ij} \neq \phi$

A_{ij}

2. A_{ij} must be consisted of optimal sols for its subproblem.

$S_{ij} \neq \phi$



$A_{ij} \cup A_{ik} \cup A_{kj} \cup \{a_k\}$ ②

3. Why it NOT = ∅

① A_{ij} is optimal

② NOC

✗ cannot coexist

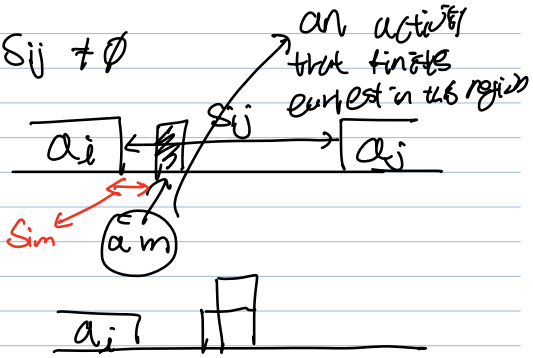
$A_{ij} \cup A_{ik} \cup A_{kj} \cup \{a_k\}$

① X
② X
③ X X

Greedy choice property Thm. 1 any $S_{ij} \neq \emptyset$

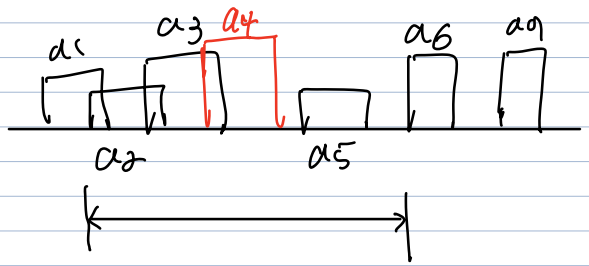
① $S_{im} \neq \emptyset$

② At least an opt sol for S_{ij} includes a_m as a member



a_1 $f_1 \leq f_2 \leq f_3$

Ex



$S_{16} = \{a_3, a_4, a_5\}$

$\{a_3, a_5\}$ 2

$\{a_4, a_5\}$

11/1

Ch16

Activity Selection problem

Th16.1

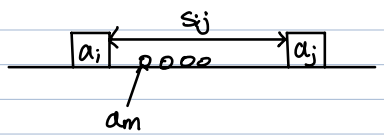
Greedy choice property
- a criterion
- → | → | → | → | →
- myopic selection

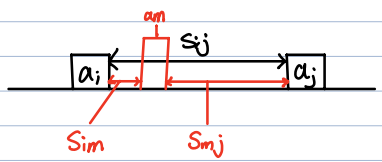
① $S_{im} \neq \emptyset$

② At least one an optimal solution for S_{ij} → ① # of S_{ij} ② $\{ \quad \}$
- includes a_m

Any Non-empty S_{ij} → a_m

$S_{ij} = \{ \quad \} \neq \emptyset$

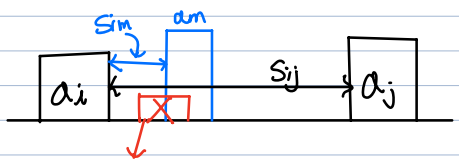




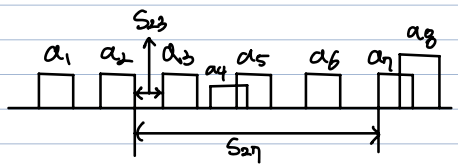
Prove by contradiction

① $S_{im} \neq \emptyset$

Assume that $S_{im} \neq \emptyset$



Ex



$S_{27} = \{a_3, a_4, a_5, a_6\}$

$a_m = a_3$

$\{a_3, a_4, a_6\}$, $\{a_3, a_5, a_6\}$ → optimal soluti

② At least one an optimal solution for S_{ij}

An optimal sol = $A = \{ \quad \}$

(1) $a_m \in A$ ✓ → $\{ \quad \}$ am이 A에 set으로 당연히 감.

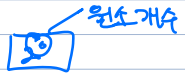
(2) $a_m \in A$

$A = \{ \quad \} \Rightarrow A' = \{a_m \quad \}$

★ 이러한 증명법

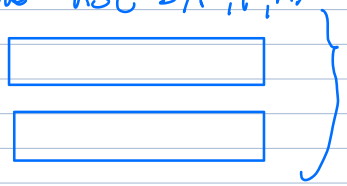
① compatible

② max

Runtime 

한번씩만 스캔해야 하므로 O(n)

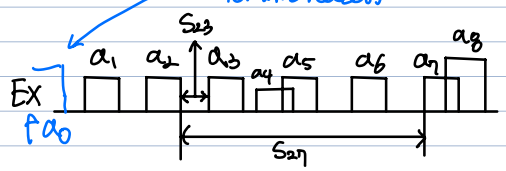
Recursive-AS(S, f, i, n) ← # of activities



recursive-AS(S, f, 0, 8)

$a_1, \ldots a_n$

a_0 → we need this for this reason



$S_1 > f_0$

$S_1 < f_0$ $S_m < f_k$

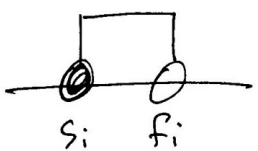
버려야한다.

an activity selection problem

input: $a_1, a_2, \cdots, a_n$ $\quad n \geq 1$

Each a_i is given by

$[s_i, f_i)$ half fly-open interval



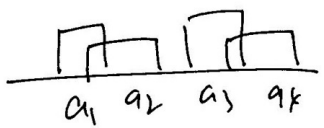
sorted in a non-decreasing order in terms of finish times

i.e. $f_1 \leq f_2 \leq \cdots \leq f_n$

output: max # of compatible activities

= non-overlapping

an instance



2 $\{a_1, a_3\}$

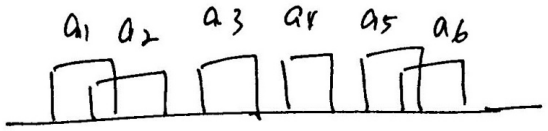
$\{a_1, a_4\}$

$\{a_2, a_3\}$

$\{a_2, a_4\}$

optimal substructure.

let $S_{ij} = \{ a_k \mid f_i \le s_k < f_k \le s_j \}$

Ex 

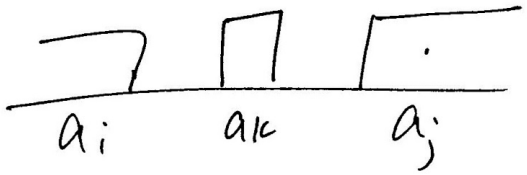
$S_{12} = \emptyset$ $S_{14} = \{a_3\}$

$S_{25} = \{a_3, a_4\}$ $S_{36} = \{a_4\}$

Consider a non-empty S_{ij}

a subproblem

let an optimal solution for S_{ij} be A_{ij}

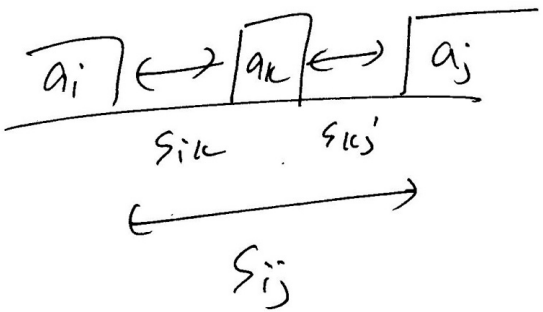


Since ~~A_{ij}~~ S_{ij} is not empty, at least one activity exists. - call it a_k.

Claim $A_{ij} = \{a_k\} \cup A_{ik} \cup A_{kj}$

optimal solution for S_{ik}

optimal solution for S_{kj}



What if NOT ?

That is. ① A_{ij} : an opt. sol for S_{ij}

② NOT (A_{ik} is an optimal sol. for S_{ik} AND A_{kj} is an opt sol for S_{kj})

Contradiction. Why ?

NOT (A_{ik} is an optimal sol for S_{ik} AND A_{kj} is an optimal sol for S_{kj}) ≡ (A_{ik} is not an optimal sol for S_{ik} OR A_{kj} is not an optimal sol for S_{kj})

But Since A_{ij} is an optimal sol for S_{ij}

→ 3 possibilities.

1. A_{ij} is an optimal sol for S_{ij}

A_{ik} is NOT an optimal sol for S_{ik}

A_{kj} is an optimal sol for S_{kj}

cannot co-exist!

→ We can have a better sol for S_{ik}. Using this sol we can have a better sol than A_{ij}

But A_{ij} is an optimal solution, so we can NOT have a better sol than A_{ij}. Contradiction

2. A_{ij} is an optimal sol for S_{ij}

A_{ik} is an opt sol for S_{ik}

A_{kj} is NOT an opt sol for S_{kj}

→ Contradiction.

3. A_{ij} is an optimal sol for S_{ij}

A_{ik} is ~~an opt~~ NOT an opt sol for S_{ik}

A_{kj} is NOT an opt sol for S_{kj}

→ Contradiction

Optimal substrucure property – an optimal solution consists of optimal solutions of subproblems

Since the activity selection problem possesses the optimal substucture property, it is POSSIBLE to solve the problem by solving smaller problems and composing optimal solutions of these subproblems to come up with an optimal solution of the entire problem

In other words, a bottom-up approach CAN be used to solve the activity selection problem [**]

BUT, the activity selection problem possesses an additional structural property – Greedy choice property ? We can use a BETTER approach than [**]

Greedy choice property – an optimal solution of the problem can be obtained by repeatedly selecting “what looks best” each time a selection is made [myopic selection is repeated]

If an optimization problem possesses

(1) Optimal substructure property, and at the same time
(2) Greedy choice property

then, a greedy algorithm works!

A greedy algorithm is an algorithm that repeatedly selects what looks best each time a selection is made.

- We need a criterion or criteria for selection
- In principle, if a problem is solvable by a greedy algorithm, then the problem can be solvable by dynamic programming (in this case, a greedy algorithm is BETTER than dynamic programming based algorithm]

Greedy choice property of the activity selection problem – theorem 16.1

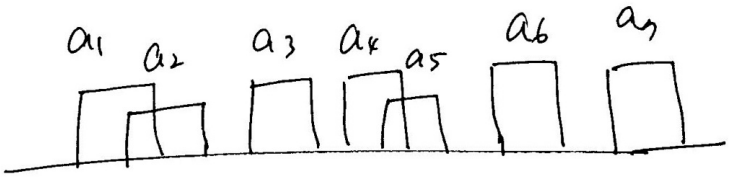
Consider any nonempty subproblem S_{ij}, and let a_m be the activity in S_{ij} with the earliest finish time:

$$f_m = \min\{f_k : a_k \in S_{ij}\} .$$

Then

1. Activity a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}.
2. The subproblem S_{im} is empty, so that choosing a_m leaves the subproblem S_{mj} as the only one that may be nonempty.

1. Ex



$S_{26} = \{a_3, a_4, a_5\}$

max # of non-overlapping activities = 2

$\{a_3, a_4\}$ $\{a_3, a_5\}$

$a_m = a_3$

$S_{37} = \{a_4, a_5, a_6\}$

max # of non-overlapping activities = 2

$\{a_4, a_6\}$, $\{a_5, a_6\}$

$a_m = a_4$

proof

2 cases

① a_m is already included in an optimal solution, moved.

② a_m is NOT in an optimal solution: A

→ Always possible to create a different optimal solution that contains a_m B

$A = \{x, \ldots\} \Rightarrow B = \{a_m \ldots\}$

max # of non-overlapping activities.

$|A| = |B|$

$x \in A$ an activity that finishes earliest in A

delete x from A -1

add a_m to A $+1$

# of activities remains the same

- non-overlapping activities

why?

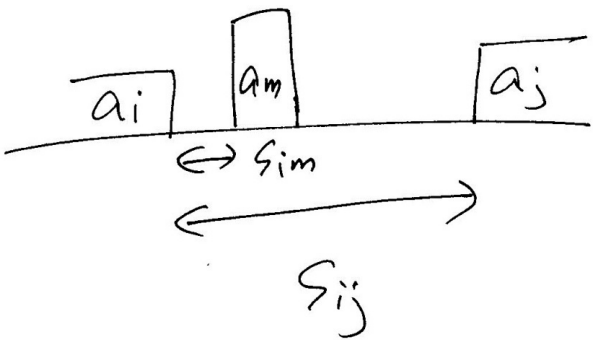
no problem with x in A.

→ no problem with a_m in A, either

2. S_{im} is empty.

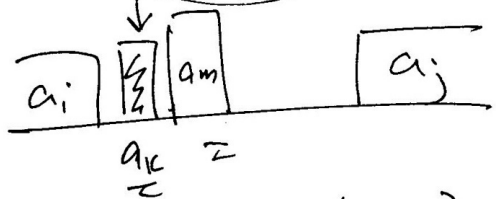
proof by contradiction.

What if NOI.



if S_{im} is not empty

then $\exists$ some activity in S_{im}



NOI possible - why?

a_m - earliest finish time!

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, i, n)
1  m ← i + 1
2  while m ≤ n and s_m < f_i        ▷ Find the first activity in S_{i,n+1}.
3        do m ← m + 1
4  if m ≤ n
5     then return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6     else return ∅
```

Line 2, 3 – “greedy” selection criterion

if
  the current activity intersects with “the most recently selected activity”
then
  skip it

Time complexity - O(n)