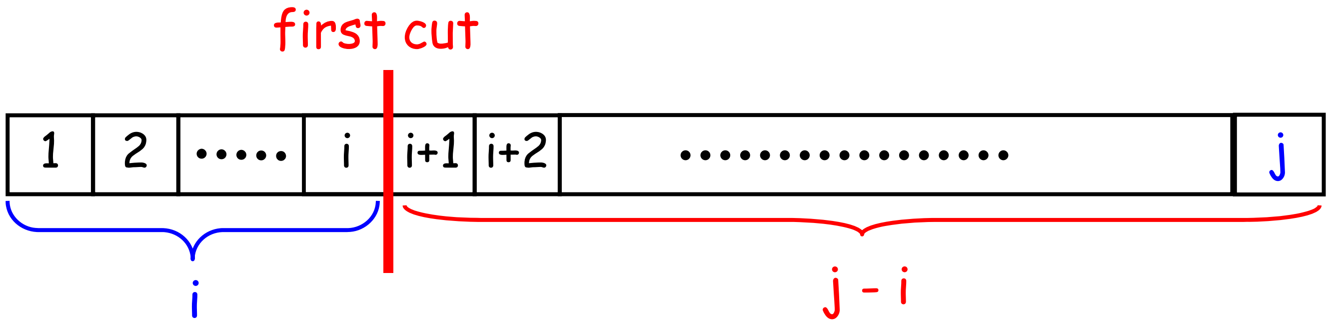


$$r[j] = \underset{1 \le i \le j}{\text{MAX}} \left\{ \underset{\text{left-part}}{p[i]} + \underset{\text{right-part}}{r[j-i]} \right\}$$

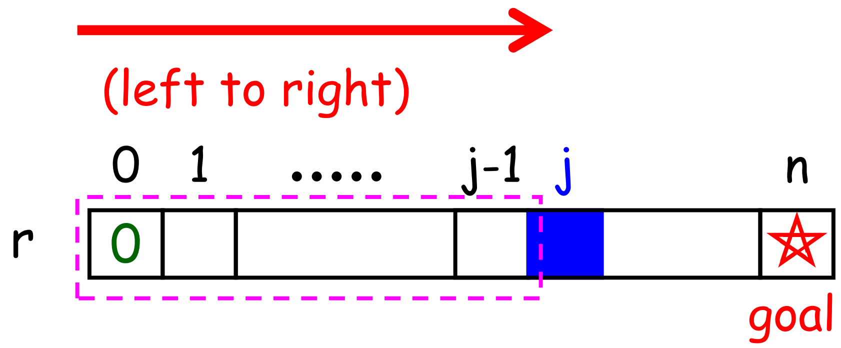
Step 3

$$r[j] = \underset{1 \le i \le j}{\text{MAX}} \left\{ p[i] + r[j-i] \right\}$$

(i) Draw a table

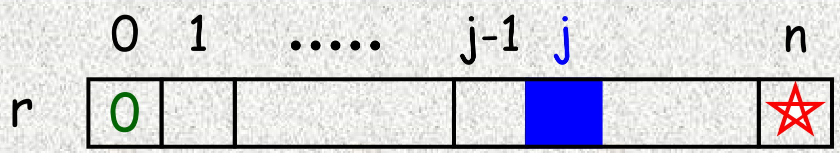
(ii) Observe the dependency

(iii) Find a good order



* r[j] needs r[0], r[1], ..., r[j-1]
(ex. r[9] needs r[0], r[1], ..., r[8])

Time complexity



$$r[j] = \underset{1 \le i \le j}{MAX} \{ p[i] + r[j] \}$$

$$\text{Time:} \quad \sum_{j=1}^{n} O(j) = \sum_{j=1}^{n} O(n) = n \times O(n) = O(n^2)$$

$$= O(1+2+...+n)$$

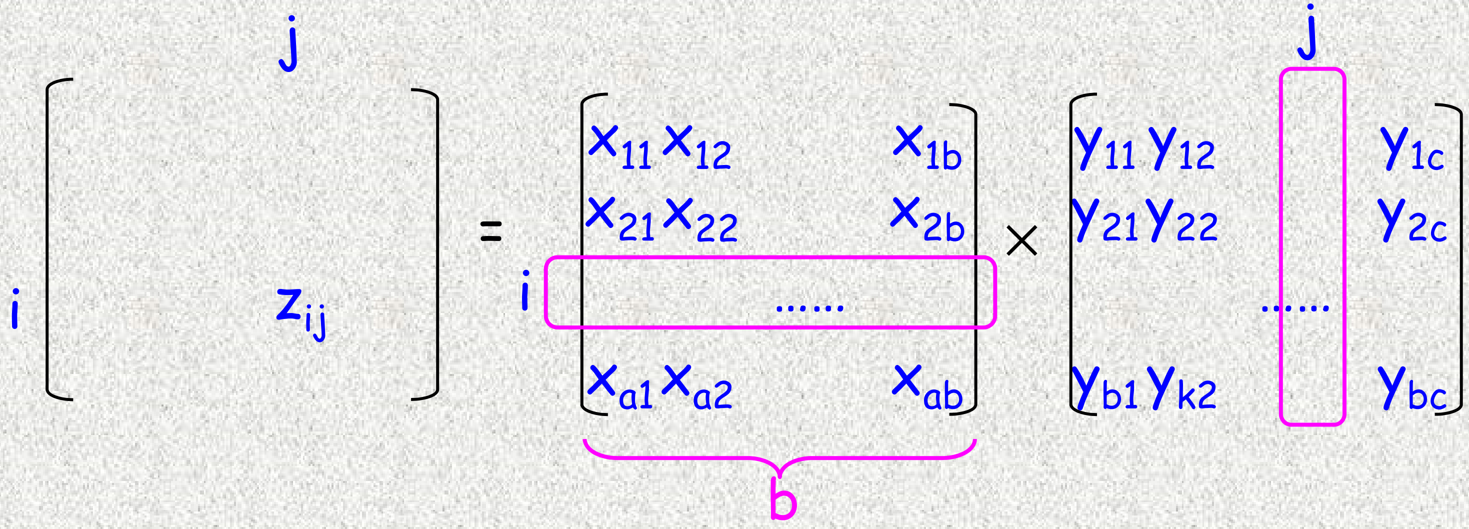
$$= O(n(n+1)/2)$$

$$= O(n^2)$$

table size

大部份情況，細算沒好處

$$Z_{a\times c} = X_{a\times b} \times Y_{b\times c}$$

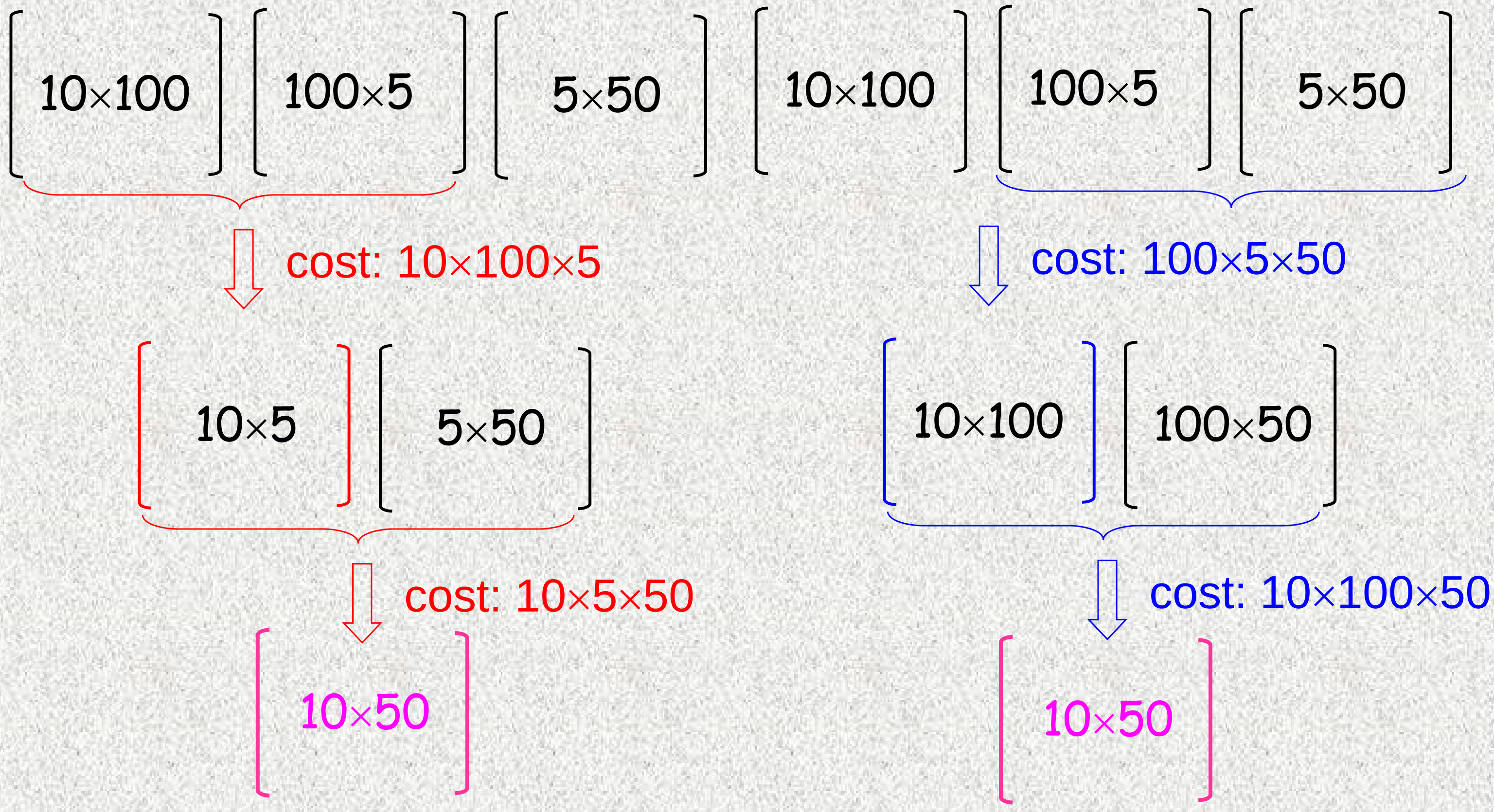


$$\begin{array}{cc} & j \\ i & \begin{bmatrix} & & \\ & z_{ij} & \\ & & \end{bmatrix} \end{array} = \begin{array}{c} \\ \\ i \\ \\ \end{array} \underbrace{\begin{bmatrix} x_{11}\,x_{12} & & x_{1b} \\ x_{21}\,x_{22} & & x_{2b} \\ & \ldots\ldots & \\ x_{a1}\,x_{a2} & & x_{ab} \end{bmatrix}}_{b} \times \begin{array}{c} \begin{array}{ccc} & j & \end{array} \\ \begin{bmatrix} y_{11}\,y_{12} & & y_{1c} \\ y_{21}\,y_{22} & & y_{2c} \\ & \ldots\ldots & \\ y_{b1}\,y_{k2} & & y_{bc} \end{bmatrix} \end{array}$$

cost of computing z_{ij} : b multiplications

cost of computing Z: abc multiplications

Note: size of Z is $a\times c$



10×100 100×5 5×50 10×100 100×5 5×50

cost: 10×100×5

cost: 100×5×50

10×5 5×50 10×100 100×50

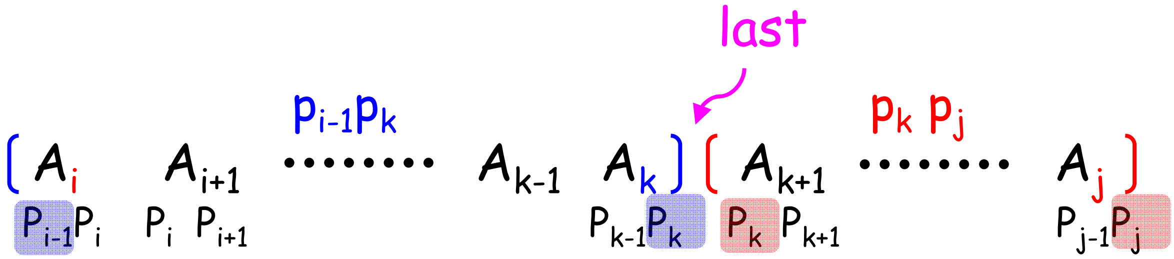
cost: 10×5×50

cost: 10×100×50

10×50 10×50

* total cost: 7500

* total cost: 75000



last

$p_{i-1}p_k$ $p_k\, p_j$

$(A_i \quad A_{i+1} \cdots\cdots A_{k-1} \quad A_k)(A_{k+1} \cdots\cdots A_j)$

$P_{i-1}P_i \quad P_i\, P_{i+1} \quad P_{k-1}P_k \quad P_k\, P_{k+1} \quad P_{j-1}P_j$

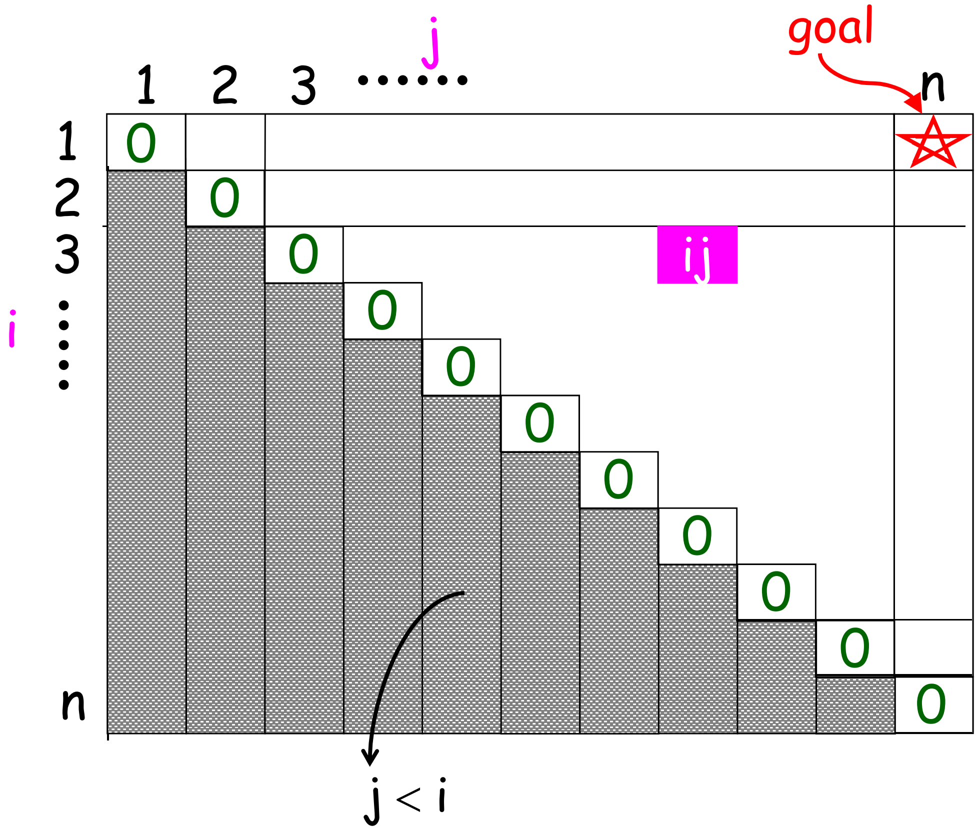
$$m_{ij} = \underset{i \le k \le j-1}{\text{MIN}} \left\{ \underset{\text{left-part}}{m_{ik}} + \underset{\text{right-part}}{m_{k+1j}} + \underset{\text{last}}{p_{i-1}p_kp_j} \right\}$$

Step 3

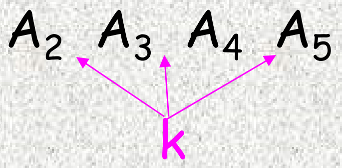
$$m_{ij} = \text{MIN}\,\{m_{ik} + m_{k+1j} + p_{i-1}p_kp_j\}$$

(i) Draw a table

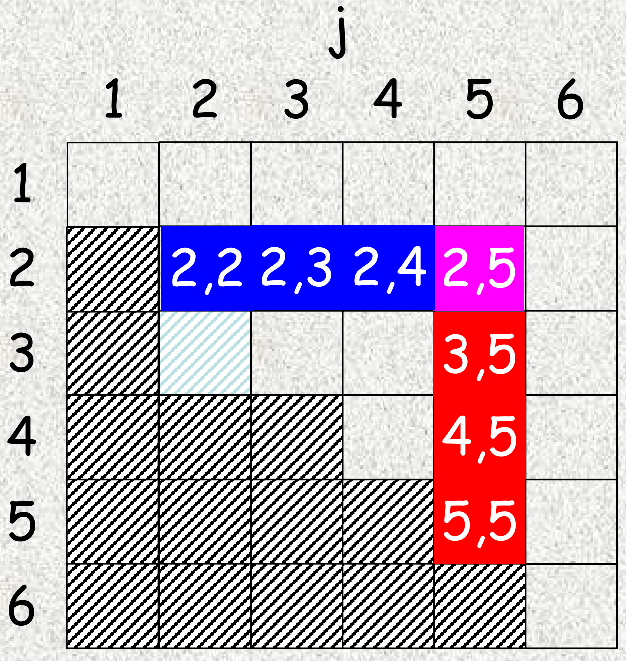
(ii) Observe the dependency



Compute m[2, 5] (See page 15-7)



$$\text{MIN}\begin{cases} m[2,2] \overset{k=2}{+} m[3,5] + p_1p_2p_5 \\ m[2,3] \overset{k=3}{+} m[4,5] + p_1p_3p_5 \\ m[2,4] \overset{k=4}{+} m[5,5] + p_1p_4p_5 \end{cases}\Bigg\} i$$



	1	2	3	4	5	6
1						
2		2,2	2,3	2,4	2,5	
3					3,5	
4					4,5	
5					5,5	
6						

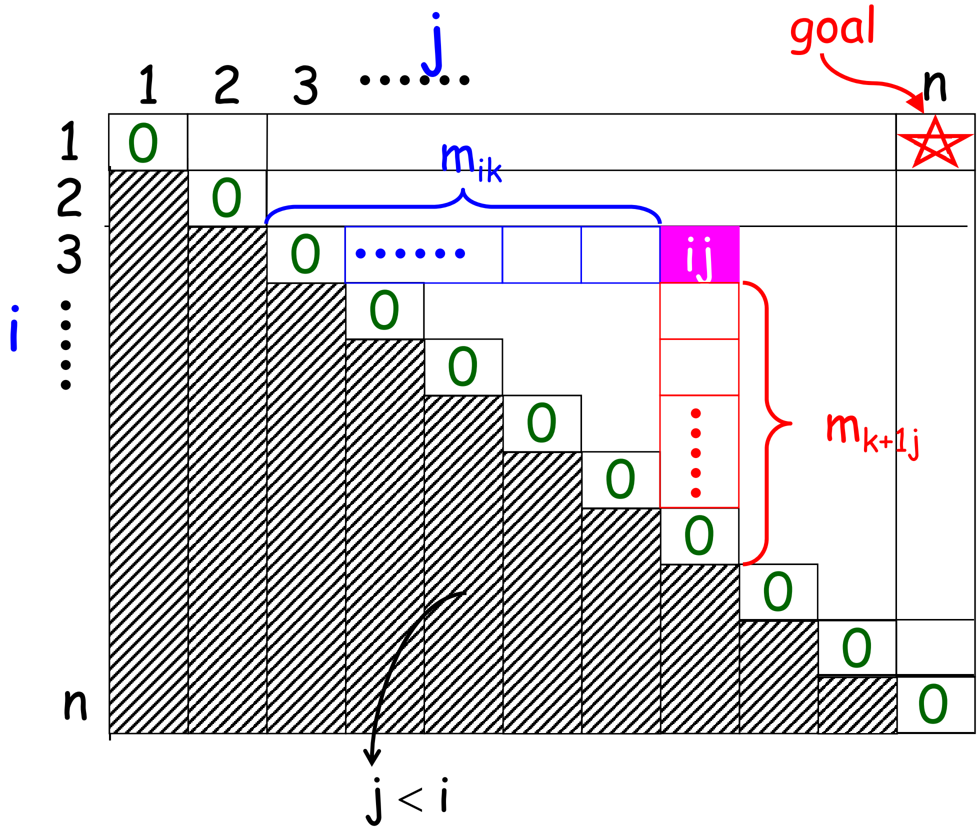
Step 3

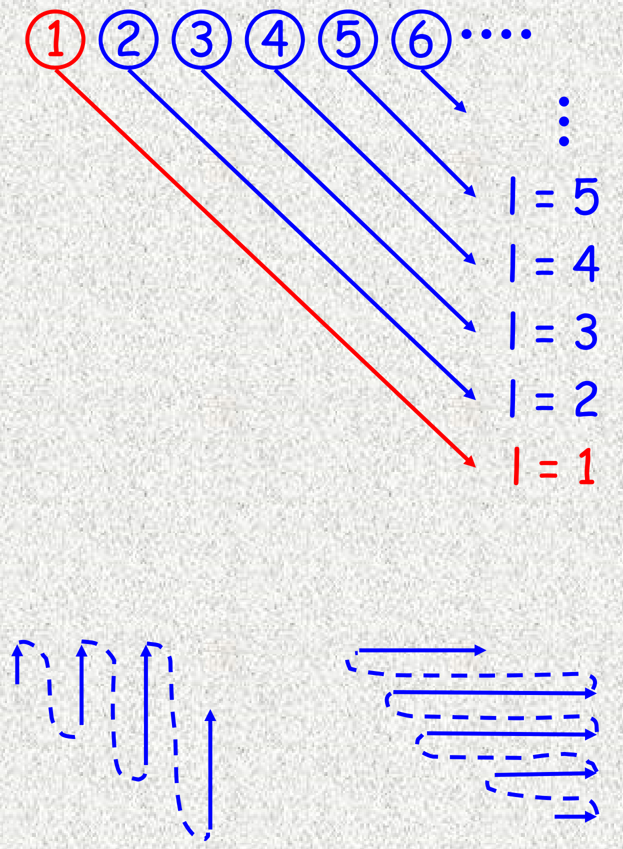
$m_{ij} = MIN \{m_{ik} + m_{k+1j} + p_{i-1}p_kp_j\}$

(i) Draw a table

(ii) Observe the dependency

(iii) Find a good order





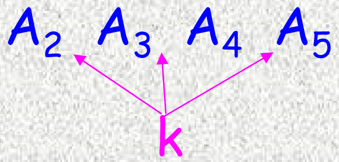
textbook

Compute m[2, 5]

(See page 15-7)

p0	p1	p2	p3	p4	p5	p6
30	35	15	5	10	20	25

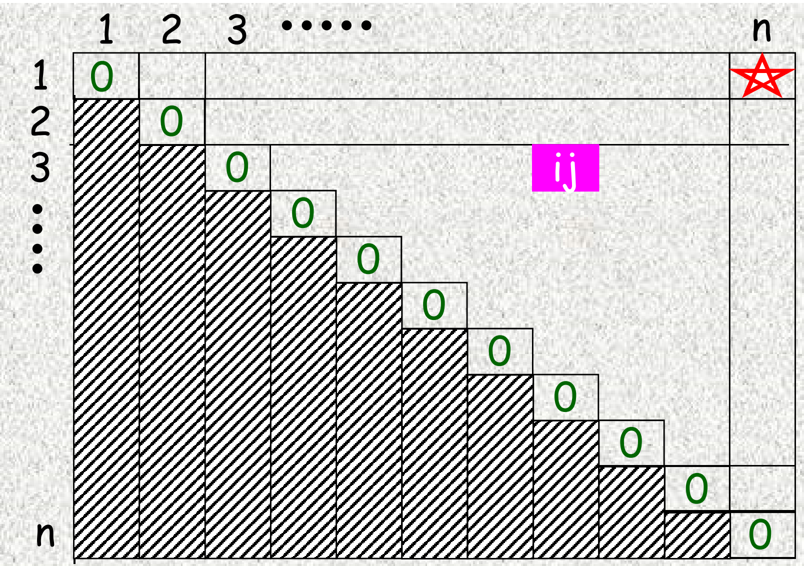
	1	2	3	4	5	6
1	0	15750	7875	9375		
2	▨	0	2625	4375	7125	
3	▨	▨	0	750	2500	
4	▨	▨	▨	0	1000	3500
5	▨	▨	▨	▨	0	5000
6	▨	▨	▨	▨	▨	0



s[2,5]

$$\text{MIN}\begin{cases} 0 + 2500 + 35\times15\times20 & (k = 2) \\ 2625 + 1000 + 35\times5\times20 & (k = 3) \\ 4375 + 0 + 35\times10\times20 & (k = 4) \end{cases} = \text{MIN}\begin{cases} 13000 \\ 7125 \\ 11375 \end{cases}$$

Time complexity

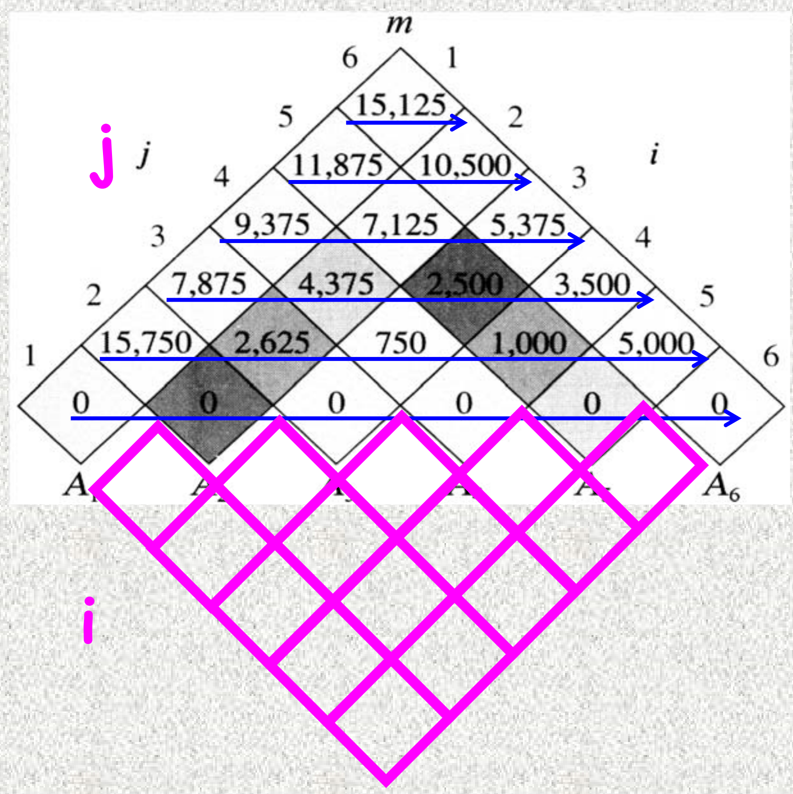


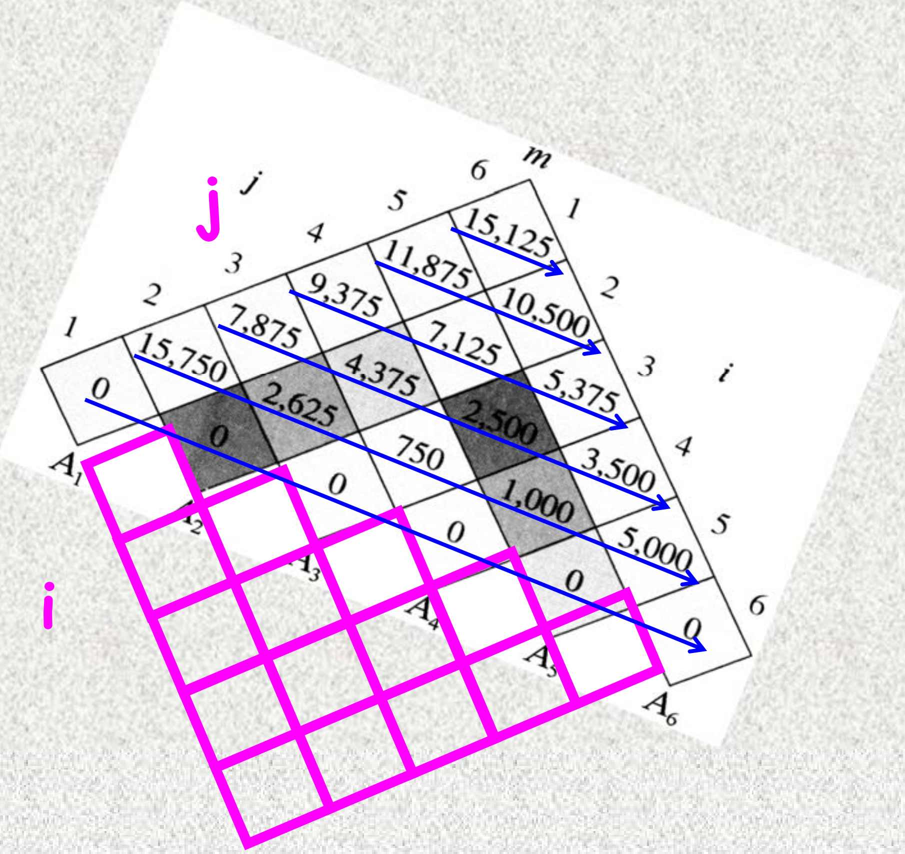
$$m_{ij} = \underset{i \le k \le j-1}{\text{MIN}} \{m_{ik} + m_{k+1j} + p_{i-1}p_kp_j\}$$

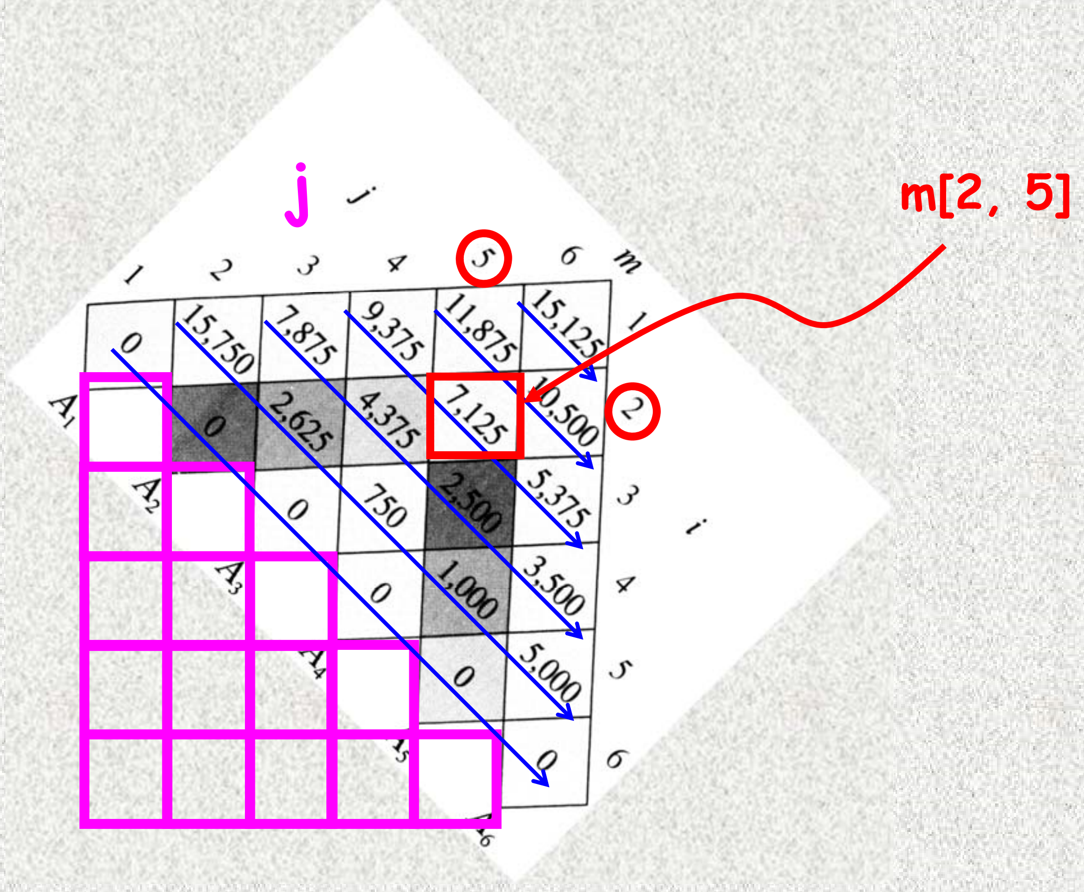
$$\text{Time: } \sum_{i,j} O(j-i) = \sum_{i,j} O(n) = O(n^2) \times O(n) = O(n^3)$$

細 算 沒 好 處

table size


m
j
i
j
i
1 2 3 4 5 6
6 5 4 3 2 1
15,125
11,875
10,500
9,375
7,125
5,375
7,875
4,375
2,500
3,500
15,750
2,625
750
1,000
5,000
0 0 0 0 0 0
A1 A2 A3 A4 A5 A6

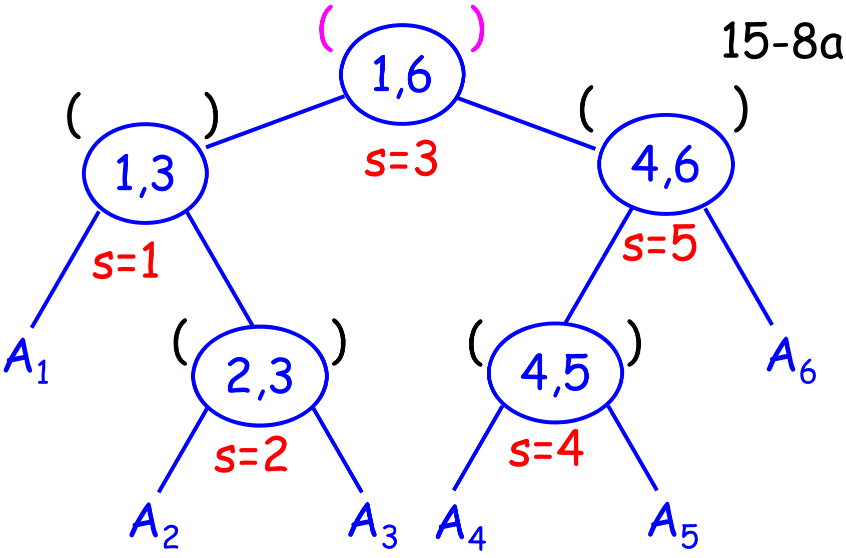

m
j
j
i
i
1 2 3 4 5 6
6 5 4 3 2 1
15,125
11,875
10,500
9,375
7,125
5,375
7,875
4,375
2,500
3,500
15,750
2,625
750
1,000
5,000
0 0 0 0 0 0
A1 A2 A3 A4 A5 A6



---

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

table s



((A_1 (A_2 A_3)) ((A_4 A_5) A_6))

a recursive procedure { one matrix: print A_i

otherwise: ① (↙② ↘③) ④

Fibonacci numbers

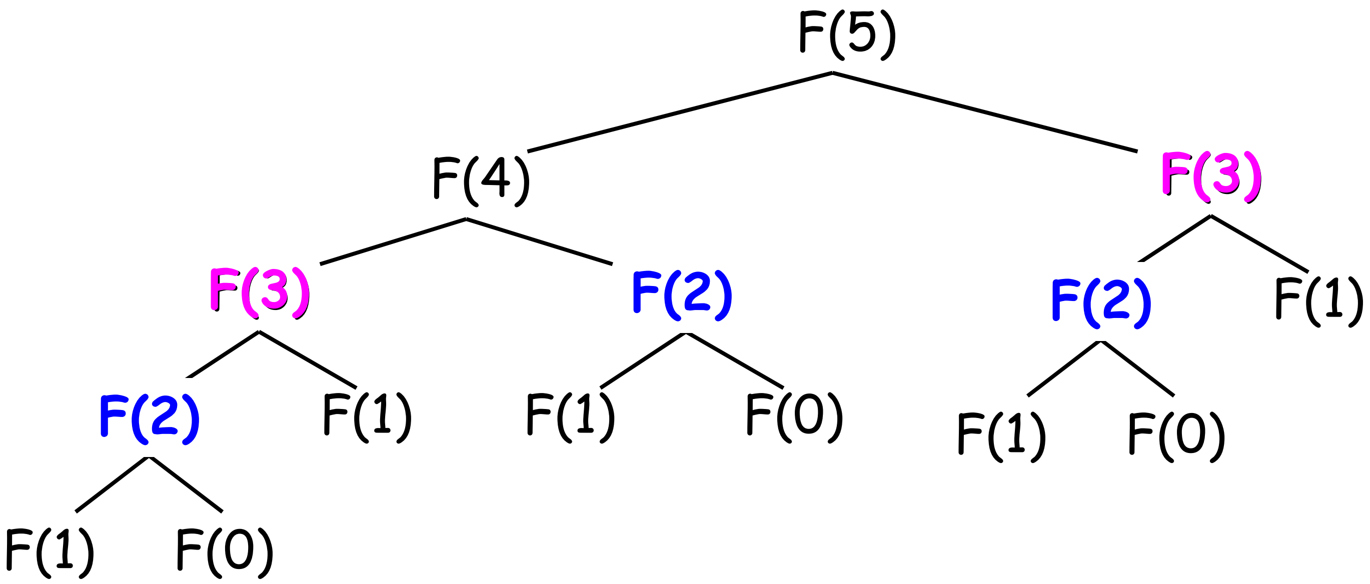
$$\begin{cases} F_0 = F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

(i) Recursive

(top-down, $O(2^n)$)

```
function F(n)
begin
    if n ≤ 1 then return 1
    else
        return F(n-1)+F(n-2);
end;
```

Top-down: $O(2^n)$ (or $O((\frac{1+\sqrt{5}}{2})^n) = O(1.618^n)$)



Fibonacci numbers

$$\begin{cases} F_0 = F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

(i) Recursive
(top-down, $O(2^n)$)

```
function F(n)
begin
    if n ≤ 1 then return 1
    else
        return F(n-1)+F(n-2);
end;
```

(ii) Tabular
(DP: bottom-up, $O(n)$)

```
F: array [0..n] of integer;

F[0] := 1; F[1] := 1;
for i:=2 to n do
    F[i] := F[i-1] + F[i-2];
return F[n];
```

Memoization : $F_n = F_{n-1} + F_{n-2}$

(top-down DP!)

	0	1	2	3	4	5
F	∞	∞	∞	∞	∞	∞

```
Mem-F(n)
    F: array [0..n] of integer;
    for i=0 to n do F[i] := ∞
    return Lookup-F(n);
```

```
Lookup-F(i)
    if F[i] ≠ ∞ then return F[i]
    ~~else~~
    if i ≤ 1 then ~~return~~ F[i] := 1
    else
        ~~return~~ F[i] := Lookup-F(i-1) +
                          Lookup-F(i-2);
    return F[i];
```

avoid recomputing

compute and save

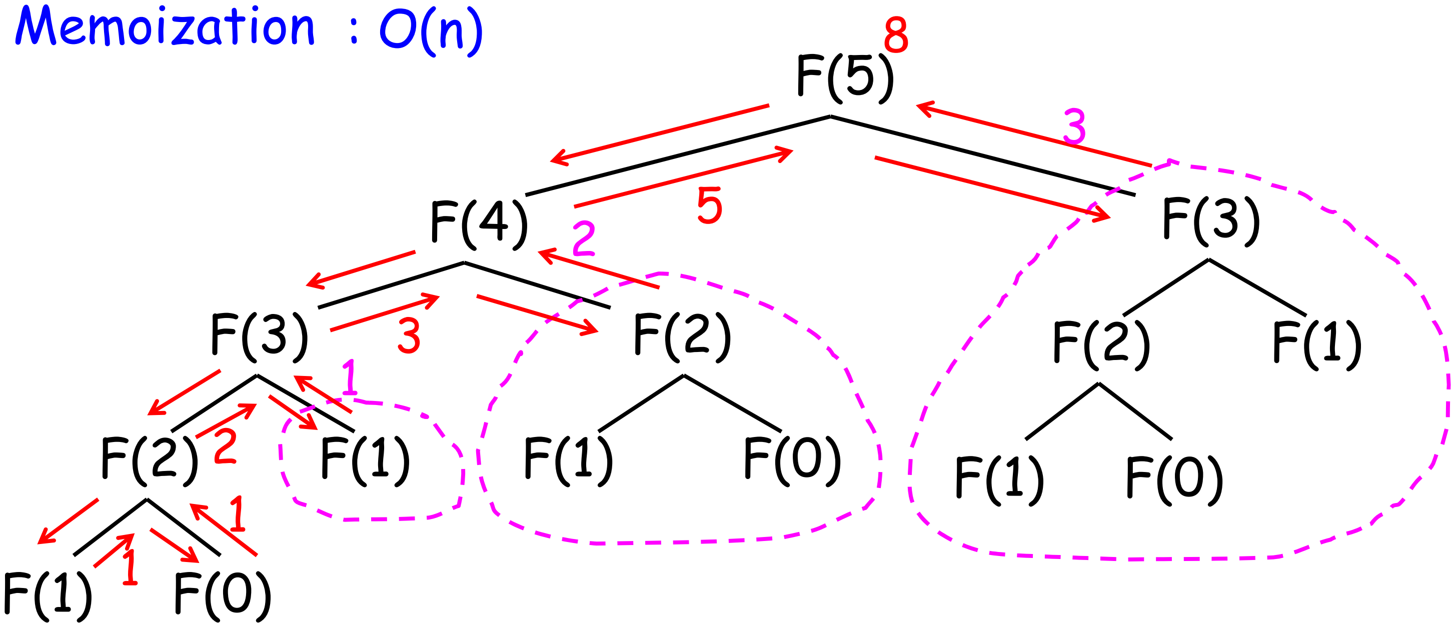
save for latter usage

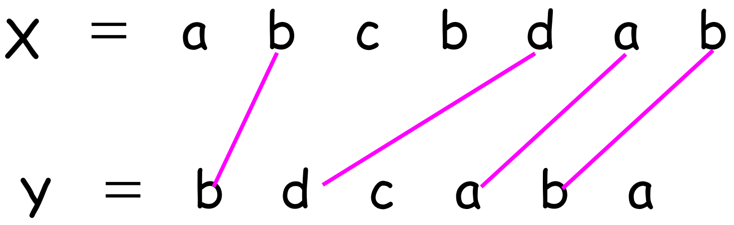
Top-down: ~~$O(2^n)$~~

+

Memoization : $O(n)$

0	1	2	3	4	5
1	1	2	3	5	8



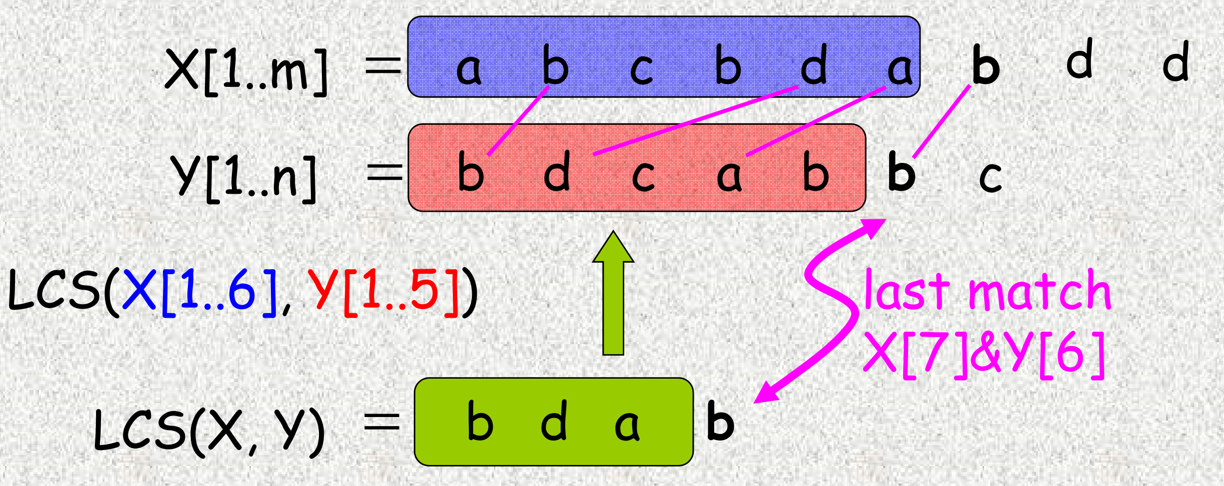


X = a b c b d a b

y = b d c a b a

LCS ⇨ max # of non-crossing matching

Optimal substructure


X[1..m] = a b c b d a **b** d d

Y[1..n] = b d c a b **b** c

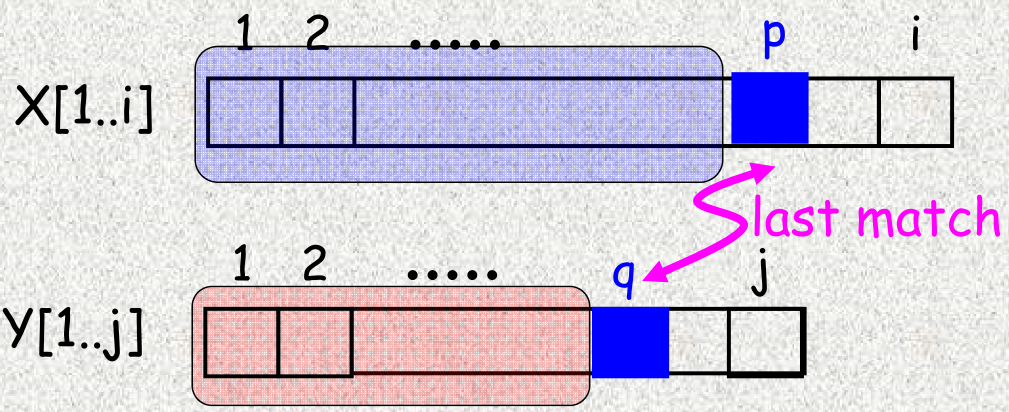
LCS(X[1..6], Y[1..5])

LCS(X, Y) = b d a **b**

last match X[7]&Y[6]

A naive DP

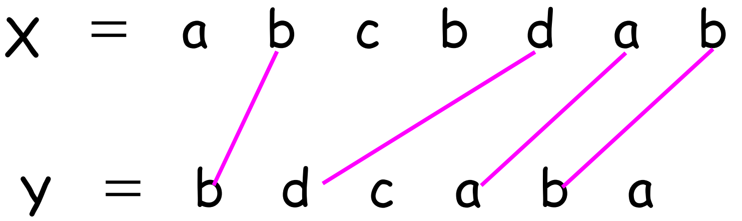
Let c[i,j] = len of LCS(X[1..i], Y[1..j])



$$c[i, j] = \max_{\substack{1 \le p \le i \\ 1 \le q \le j \\ X[p]=Y[q]}} \left\{ c[p-1, q-1] + 1 \right\}$$

last match

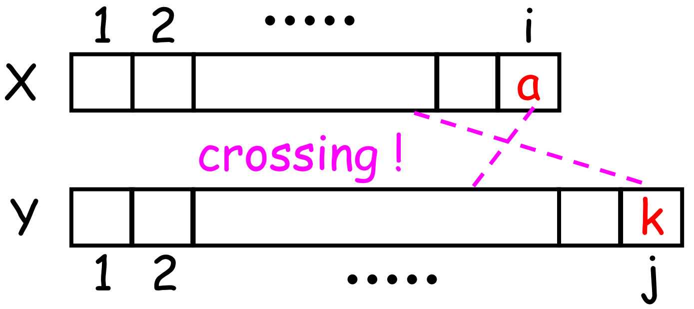
Time: $O(n^4)$



LCS ⇨ max # of non-crossing matching

Case 1. $x_i = y_j$ (discussed latter)

Case 2. $x_i \neq y_j$

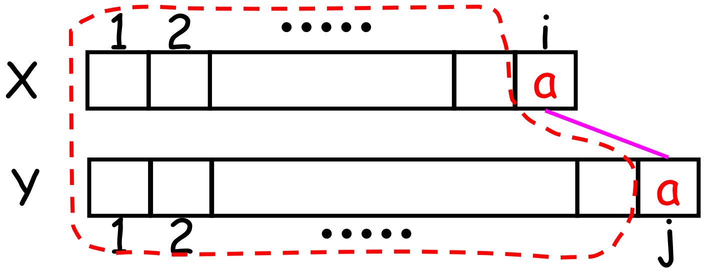


At least one of x_i and y_j can be discarded !

Let c[i,j] = LCS(X[1..i], Y[1..j])

$$c[i, j] = \text{MAX}\left\{ \begin{array}{l} \overset{\text{discard } y_j}{c[i, j-1]} \\ \underset{\text{discard } x_i}{c[i-1, j]} \end{array} \right\}$$

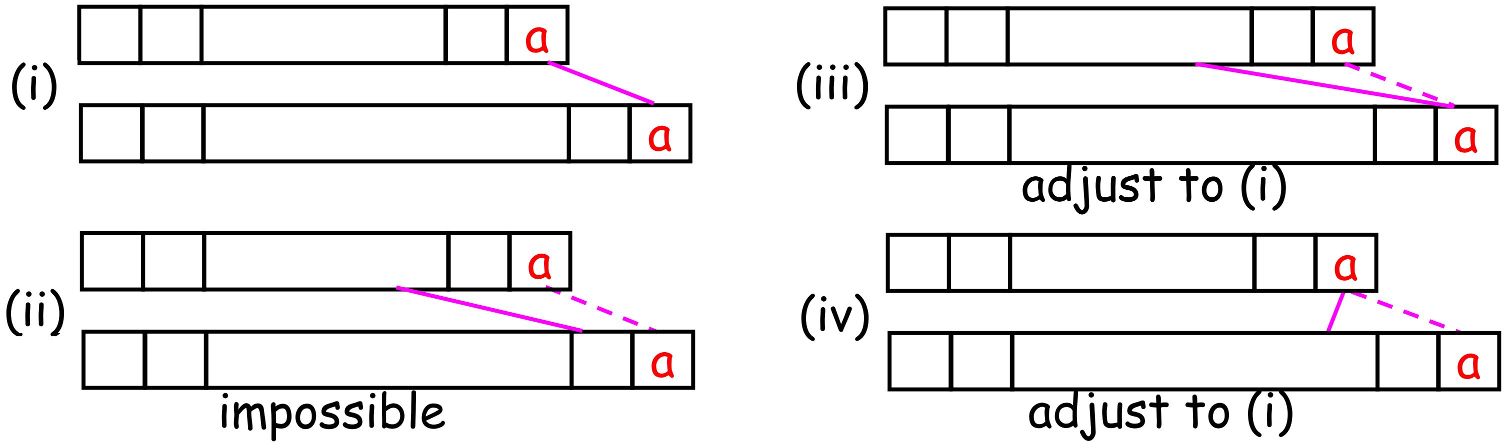
Case 1. $x_i = y_j$



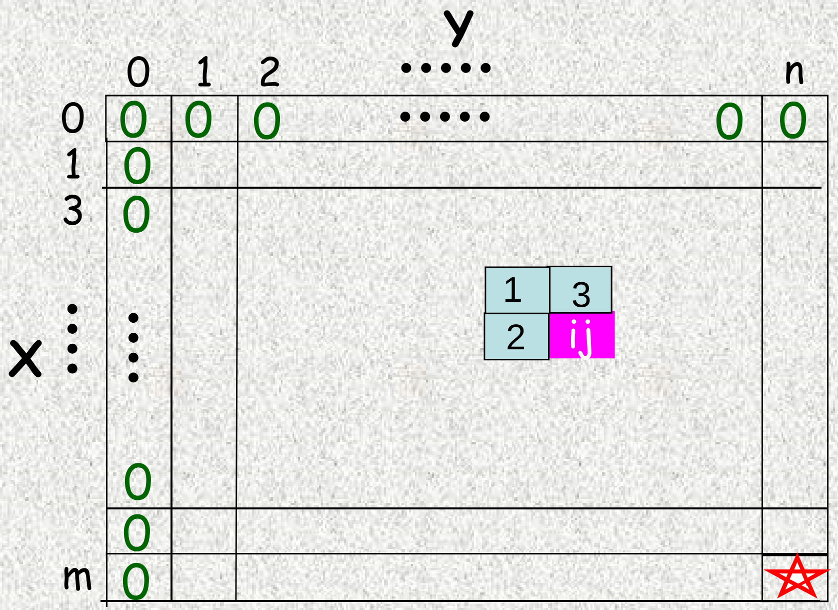
Match the tails !

$c[i, j] = c[i-1, j-1] + 1$

last matching of an optimal solution: 4 cases



Dependency and Time complexity



Time:

$\sum_{i,j} O(1) = O(mn) \times O(1) = O(mn)$

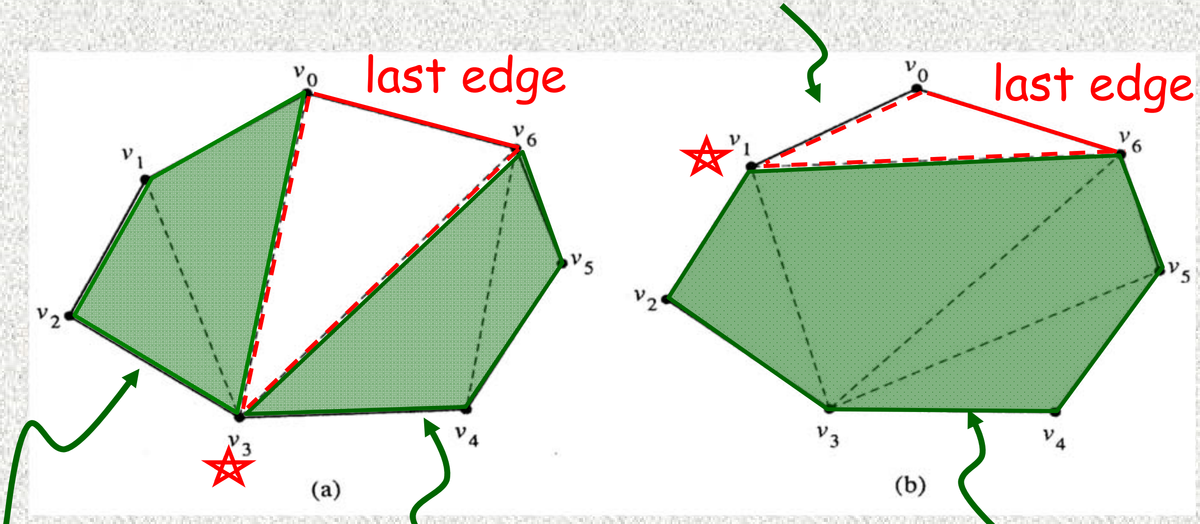
table size

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \overset{\textcircled{1}}{c[i-1, j-1]} + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{\underset{\textcircled{2}}{c[i, j-1]}, \underset{\textcircled{3}}{c[i-1, j]}\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

$(0 \leq i \leq m, 0 \leq j \leq n)$

Optimal substructure

Optimal for $P(v_0, v_1)$ (empty)



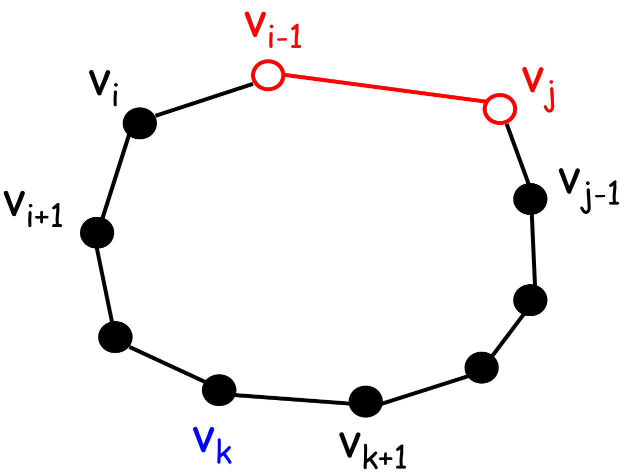
Optimal for $P(v_0, v_1, v_2, v_3)$

Optimal for $P(v_3, v_4, v_5, v_6)$

Optimal for $P(v_1, v_2, v_3, v_4, v_5, v_6)$

t[i,j] : optimal triangulation of P(v_{i-1}, v_i, …, v_j)

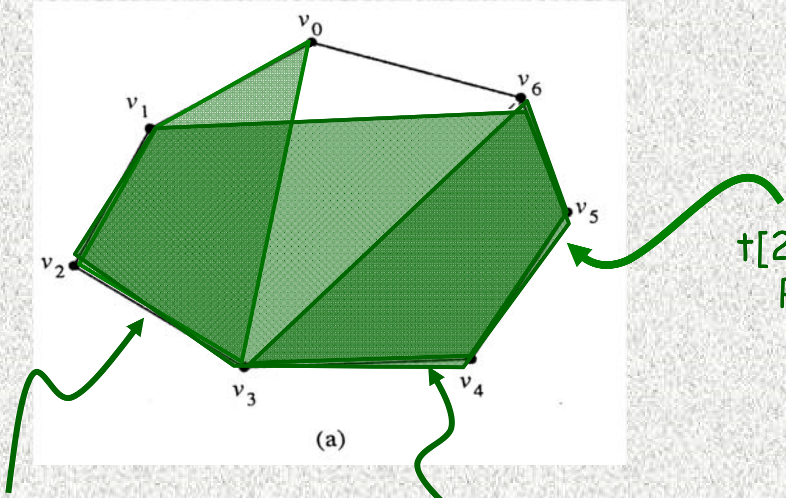
☆ 包含 v_{i-1}



---

t[i,j] : optimal triangulation of P(v_{i-1}, v_i, …, v_j)

Optimal cost for the whole input: t[1, 6] (or t[1, n-1])



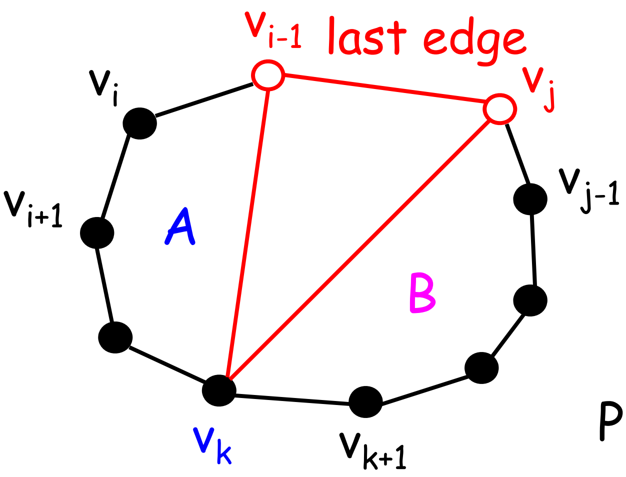
t[2, 6]: optimal cost for P(v_1, v_2, v_3, v_4 , v_5, v_6)

t[1, 3]: optimal cost for P(v_0, v_1, v_2, v_3)

t[4, 6]: optimal cost for P(v_3, v_4, v_5, v_6)

t[i,j] : optimal triangulation of $P(v_{i-1}, v_i, \ldots, v_j)$

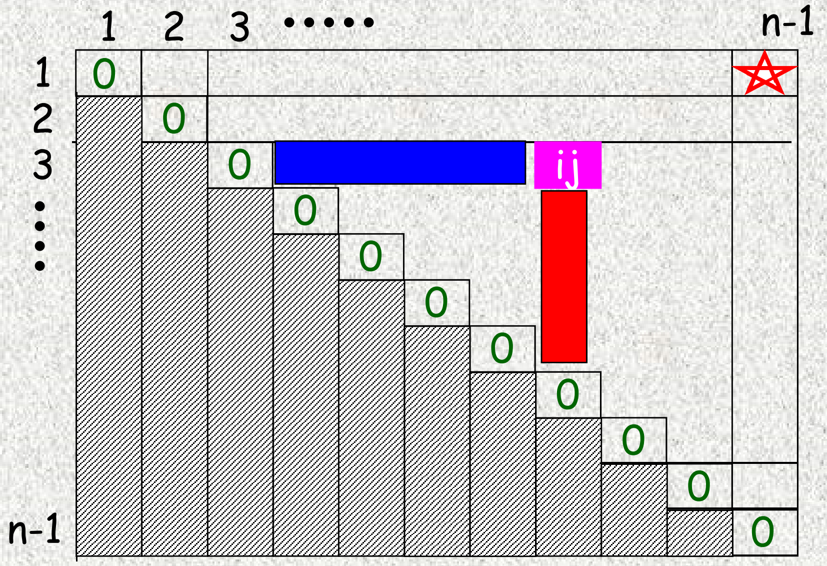
☆ 包 含 v_{i-1}



$$\overset{A}{P(v_{i-1},\ldots,v_k)} + \Delta v_{i-1}v_kv_j + \overset{B}{P(v_k,\ldots,v_j)}$$

$$t[i,j] = \underset{i \le k \le j-1}{\text{MIN}} \{ t[i, k] + w(\Delta v_{i-1}v_kv_j) + t[k+1, j] \}$$

Time complexity



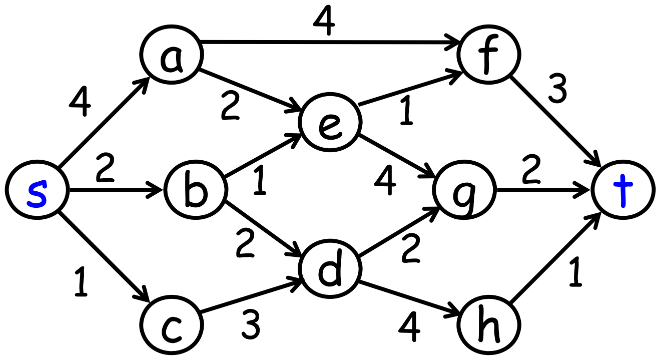
$$t_{ij} = \underset{i \le k \le j-1}{\text{MIN}} \{t_{ik} + t_{k+1j} + w(v_{i-1}v_kv_j)\}$$

Similar to Matrix-Chain

$$m_{ij} = \underset{i \le k \le j-1}{\text{MIN}} \{m_{ik} + m_{k+1j} + p_{i-1}p_kp_j\}$$

Time: $$\sum_{i,j} O(j-i) = \sum_{i,j} O(n) = \underbrace{O(n^2)}_{\text{table size}} \times O(n) = O(n^3)$$

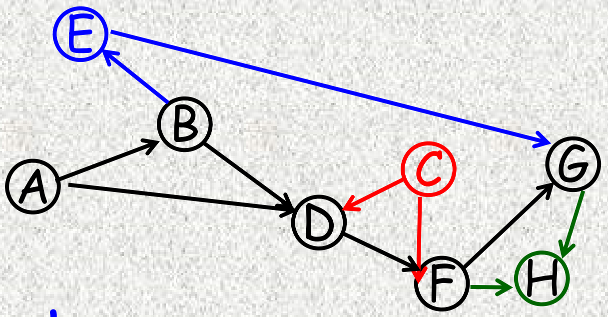
Directed Acyclic graph (multi-stage graph)



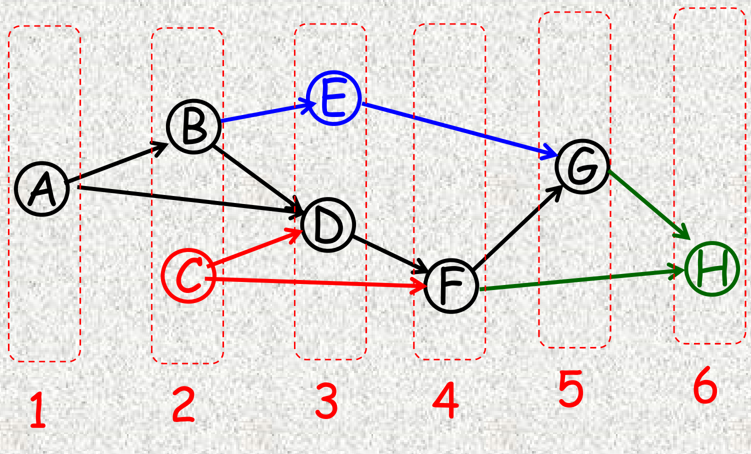
Given a DAG G = (V, E), find a shortest path from s to t

Directed Acyclic graph: directed; having no cycles

equivalent

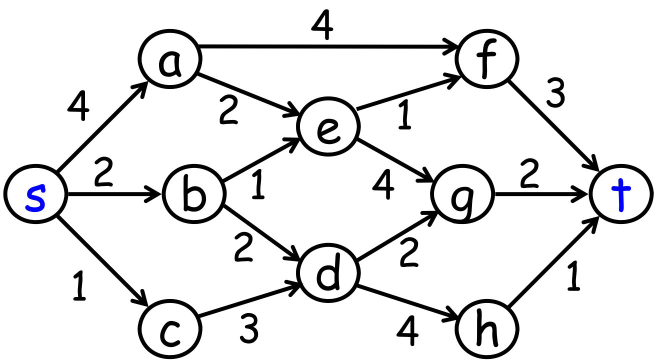


Multi-stage graph:
every edge is from a stage i to a stage j > i



topological sort (Ch 22)

Directed Acyclic graph (multi-stage graph)



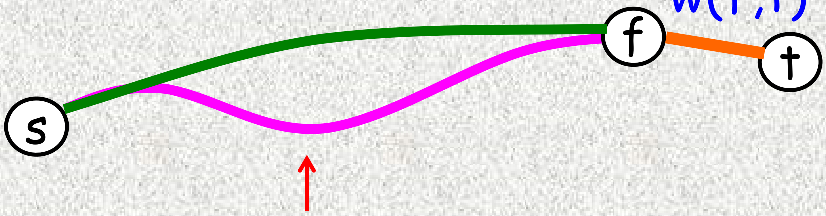
Given a DAG G = (V, E), find a shortest path from s to t

* d(v) : shortest distance from s to v

Optimal substructure

Shortest from s to t

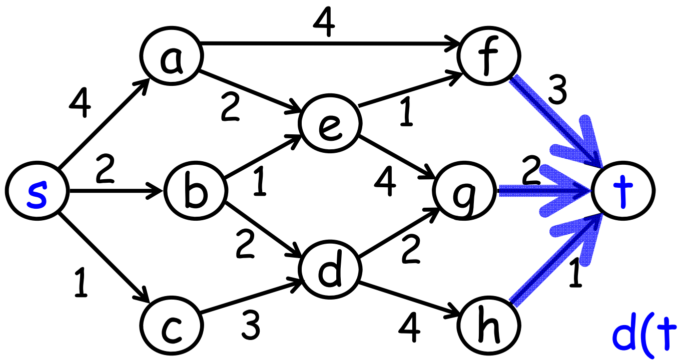
w(f,t)



Shortest from s to f

$\Rightarrow$ d(t) = d(f) + w(f,t)

Directed Acyclic graph (multi-stage graph)



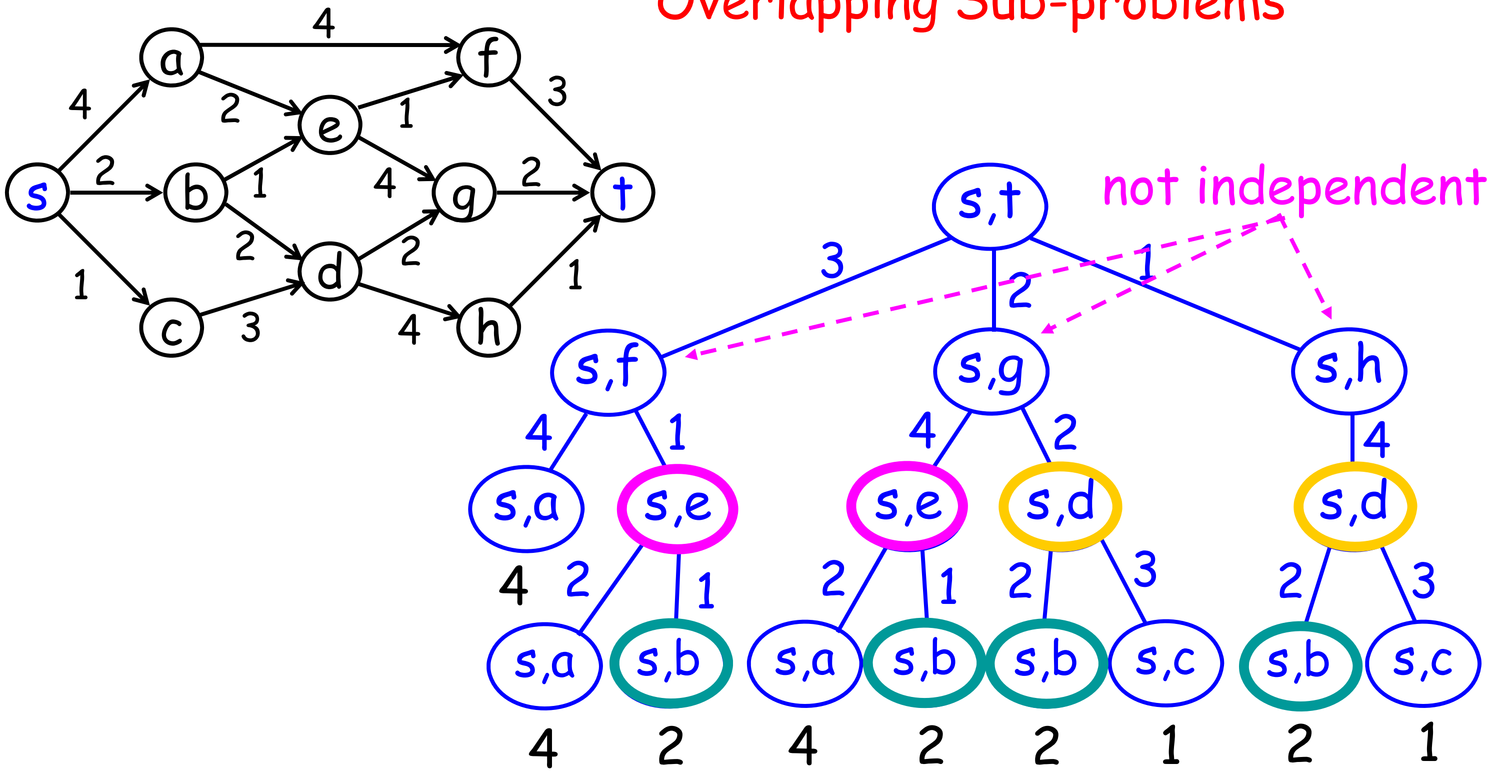
Given a DAG G = (V,E), find a shortest path from s to t

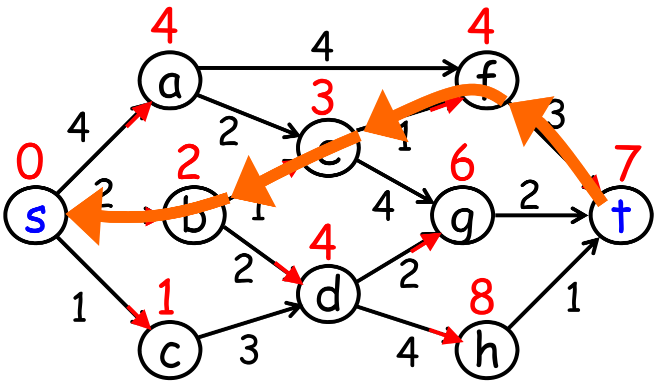
$d(t) = \text{MIN}\{ d(f)+3, d(g)+2, d(h)+1 \}$

* d(v) : shortest distance from s to v

$$\begin{cases} d(s) = 0 \\ d(v) = \underset{<x,v>\in E}{\text{MIN}}\{ d(x) + w(x,v) \} \end{cases}$$

Overlapping Sub-problems

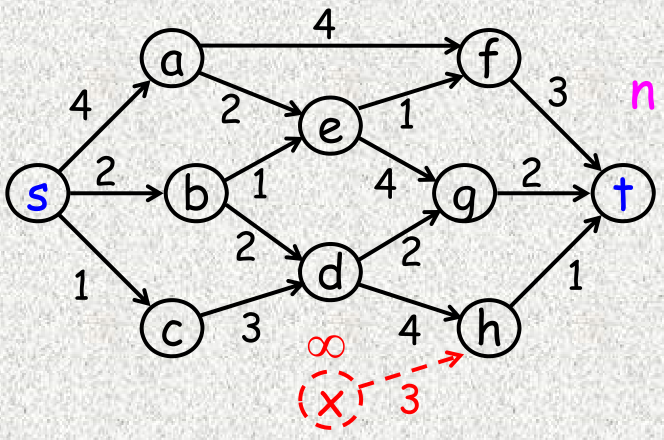




$$\begin{cases} d(s) = 0 \\ d(v) = \underset{\langle x,v\rangle \in E}{\mathrm{MIN}} \{ d(x) + w(x,v) \} \end{cases}$$

- * shortest distance d(t): DP (bottom-up, left-to-right)
  - \- a graph-shaped table
  - \- find an order: topological sort or top-down DP
- * shortest s-t path: backtracking (a simple recursive procedure)
- * O(V + E)
- * A longest path (critical path) ⇨ MIN → MAX

Time complexity



n = |V|, m = |E| (or n = V, m = E)

$$d(v) = \underset{\langle x,v\rangle \in E}{\mathrm{MIN}} \{ d(x) + w(x,v) \}$$

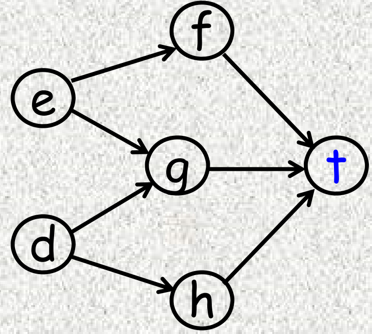
$$\text{Time: } \sum_{v} O(\text{in\_deg}(v)) = \sum_{v} O(n) = O(\overset{\text{table size}}{n}) \times O(n) = O(V^2)$$

$$\text{Time: } \sum_{v} O(\underbrace{\text{in\_deg}(v)}_{\max\{1,\ \text{in\_deg}(v)\}}) = O(\underbrace{n}_{\text{why???}} + \text{total-in-degree})$$

total-in-degree

each edge: in = +1; out = +1

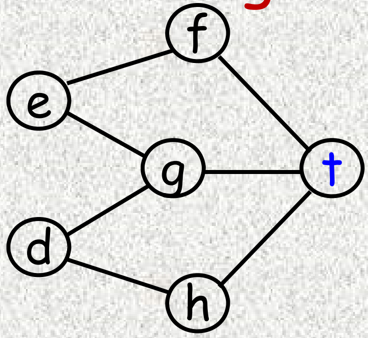
directed graph



total-in-degree = E

each edge: in = +2; out = +2

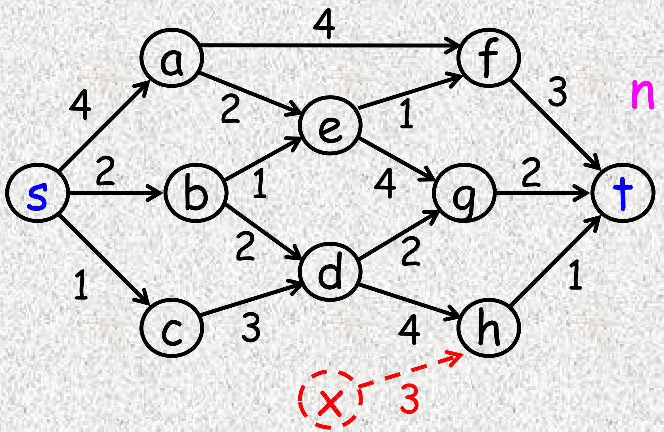
undirected graph



total-in-degree = 2E

Time complexity

directed: total-in-degree = E



n = |V|, m = |E| (or n = V, m = E)

$$d(v) = \underset{\langle x,v\rangle \in E}{\text{MIN}} \{ d(x) + w(x,v) \}$$

$$\text{Time: } \sum_{v} O(\text{in\_deg}(v)) = \sum_{v} O(n) = \overset{\text{table size}}{O(n)} \times O(n) = O(V^2)$$

$$\text{Time: } \sum_{v} O(\underset{\max\{1,\ \text{in\_deg}(v)\}}{\text{in\_deg}(v)}) = O(\underset{\text{why???}}{n} + \overset{細算有好處}{\text{total-in-degree}})$$

$$= O(n + E) = O(V + E)$$

The rod-cutting problem

$$r[j] \;=\; \underset{1 \le i \le j}{\text{MAX}} \;\{\, p[i] + r[j] \,\}$$

length i	1	2	3	4
price $p[i]$	2	5	6	9

A price table

Example: n = 4

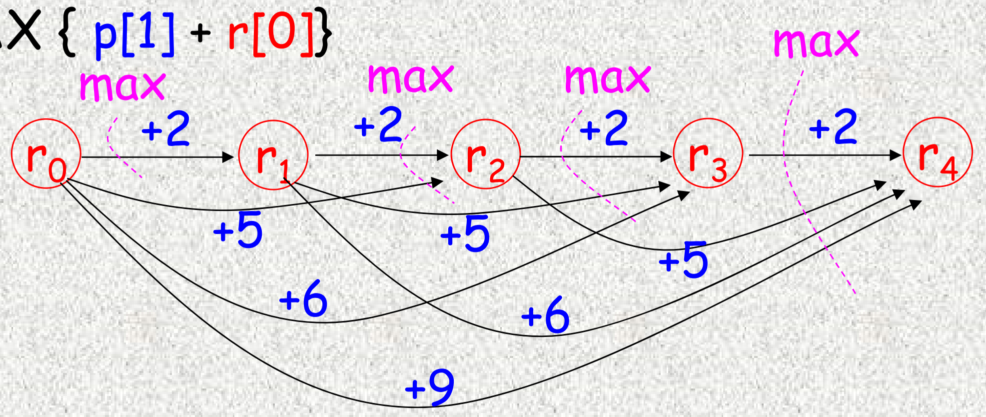
	0	1	2	3	4
r	0				☆

$r[4] = \text{MAX}\,\{\, p[1] + r[3],\; p[2] + r[2],\; p[3] + r[1],\; p[4] + r[0] \,\}$

$r[3] = \text{MAX}\,\{\, p[1] + r[2],\; p[2] + r[1],\; p[3] + r[0] \,\}$

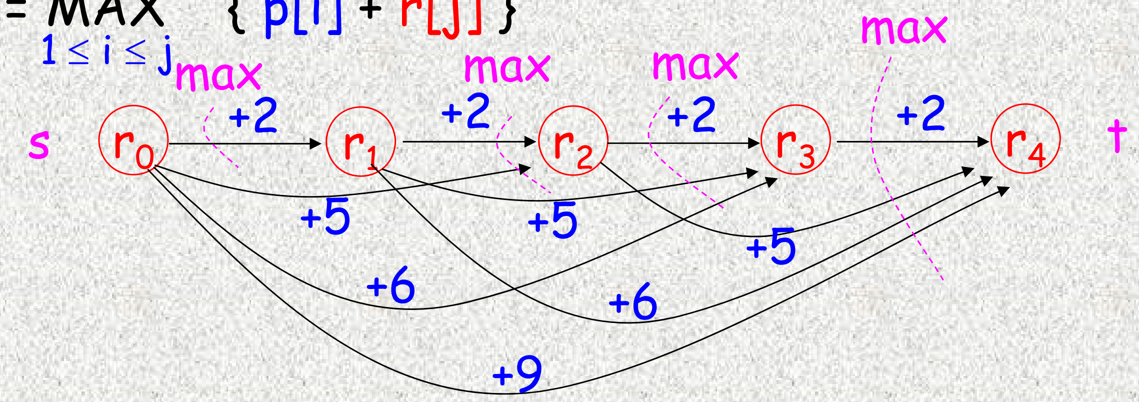
$r[2] = \text{MAX}\,\{\, p[1] + r[1],\; p[2] + r[0] \}$

$r[1] = \text{MAX}\,\{\, p[1] + r[0] \}$



The rod-cutting problem

$$r[j] \;=\; \underset{1 \le i \le j}{\text{MAX}} \;\{\, p[i] + r[j] \,\}$$



a DAG (or multistage graph)

compute r[4] = find the longest path from s to vertex t

Step 2. recurrence

Solve a problem by

Phase 1. Build a DAG (multistage graph)

Phase 2. Find a longest (shortest) path

} by DP

Step 3. fill a table

A simple exercise

$S = \{1, 3, 5, 10\}$: a set of stamps

$F(n)$: minimum # of stamps having a total of n

$n = 1 \Rightarrow \{1\}$

$n = 2 \Rightarrow \{1, 1\}$

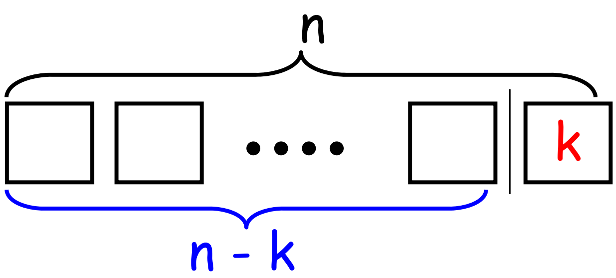
$n = 3 \Rightarrow \{3\}$

$n = 4 \Rightarrow \{1, 3\}$

$n = 9 \Rightarrow \{1, 3, 5\}$

optimal substructure

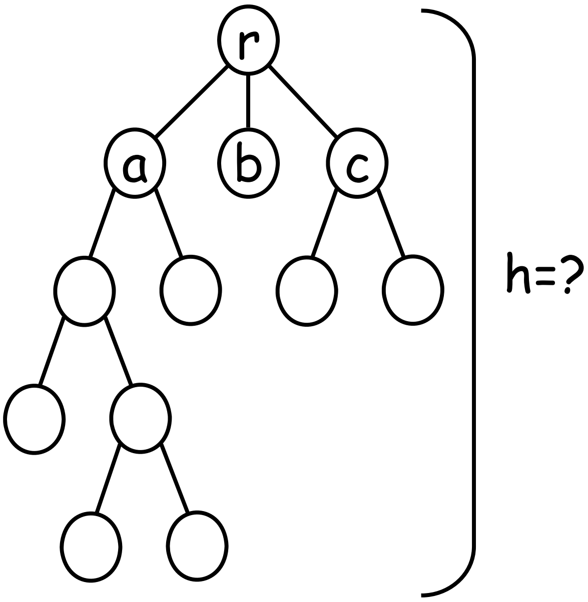
→ $F(9) = F(4) + 1$



$$F(n) = \min_{k \in S,\, k \leq n} \{ F(n-k) + 1 \}$$

* $F(0) = 0$

Dynamic Programming ?



* find a deepest leaf
  - an optimization problem

* $h(r) = \max\{ h(a), h(b), h(c) \} + 1$

* $h(v) = \max_{c \in CHILD(v)} \{ h(c) \} + 1$

$(h(v) = 0$ if v is a leaf$)$

optimal substructure

* table is tree-shaped

* no overlapping sub-problems

not need to avoid recomputing by saving answers