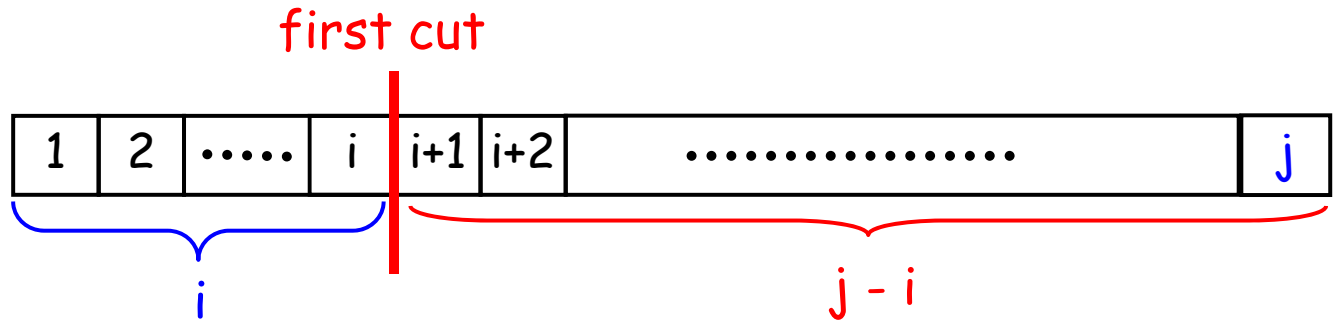


$$r[j] = \max_{1 \le i \le j} \{ p[i] + r[j-i] \}$$

left-part: $p[i]$ right-part: $r[j-i]$

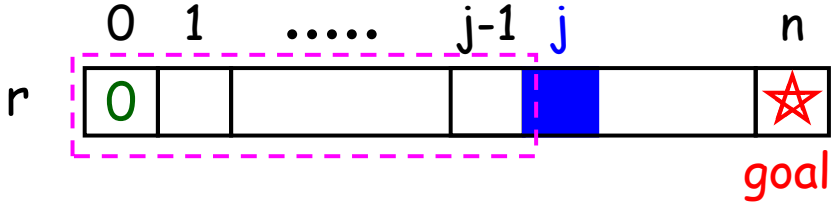
Step 3

$$r[j] = \max_{1 \le i \le j} \{ p[i] + r[j-i] \}$$

(i) Draw a table

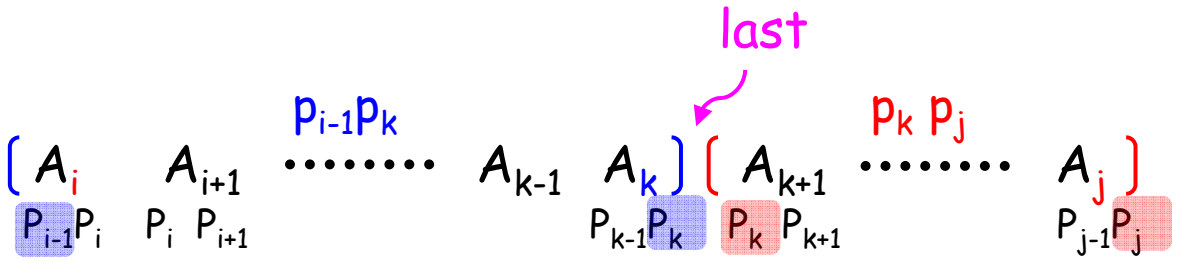
(left to right)

(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])



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

m_{ik}: left-part; m_{k+1j}: right-part; $p_{i-1}p_kp_j$: last

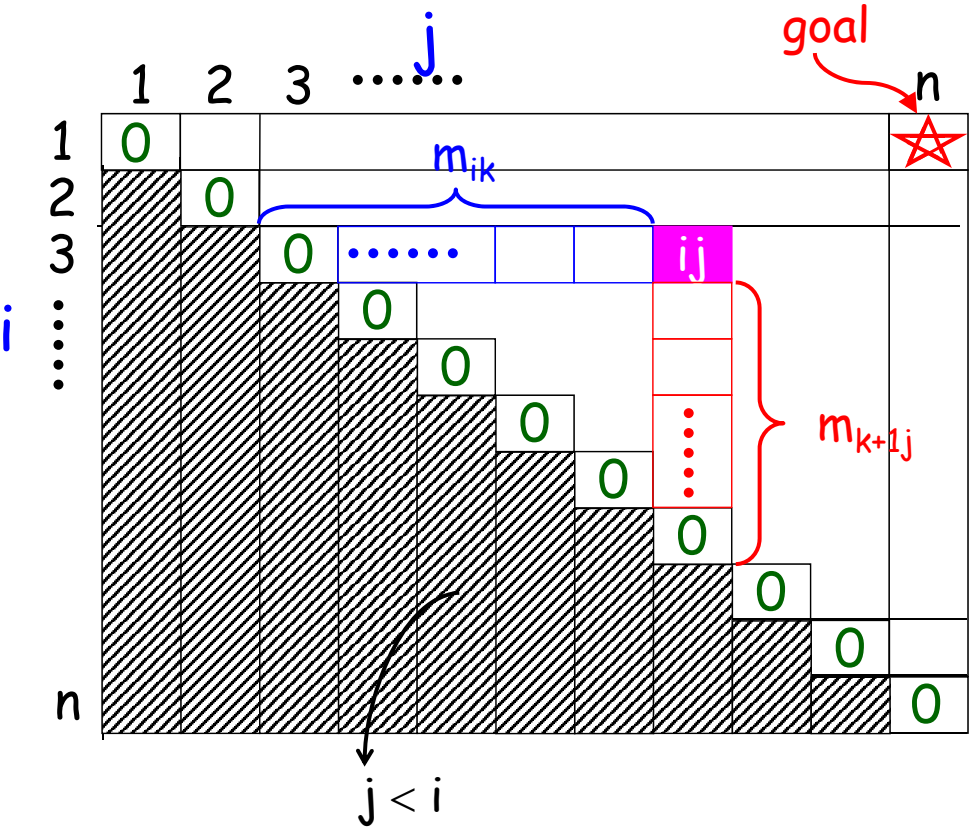
Step 3

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

(i) Draw a table

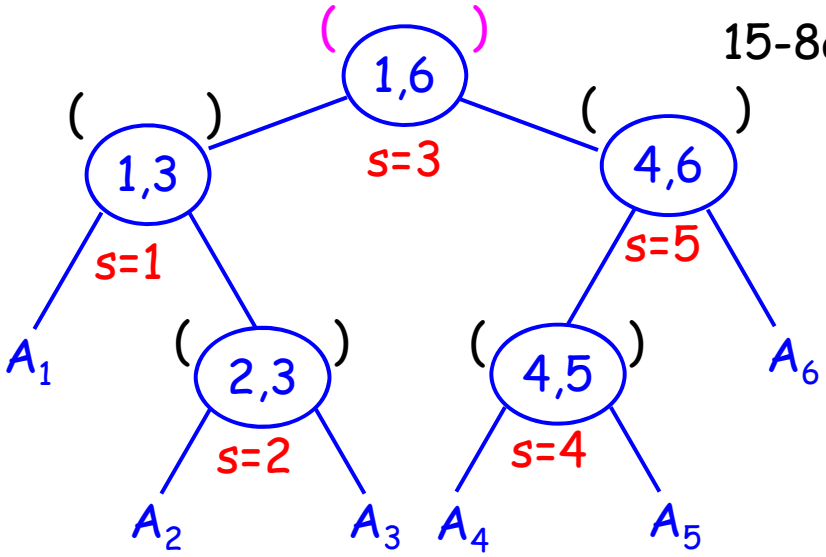
(ii) Observe the dependency

(iii) Find a good order



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



((A1 (A2 A3)) (((A4 A5) A6)))

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, $O(2^n)$)

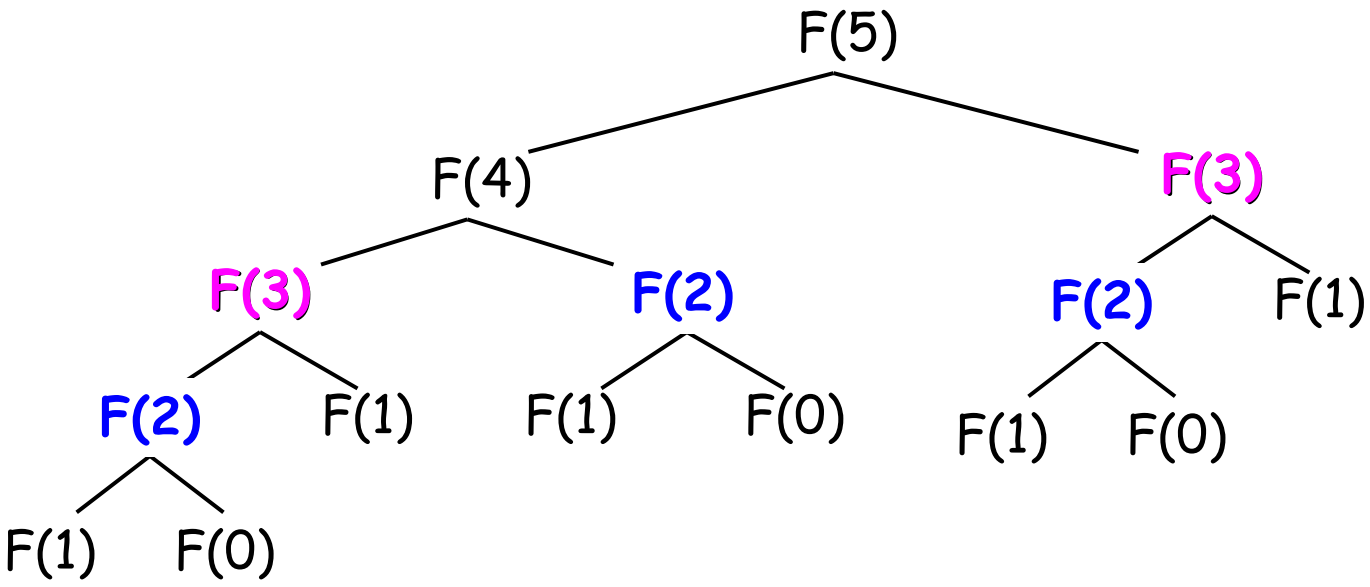
```
function F(n)
begin
   if n ≤ 1 then return 1
   else
      return F(n-1)+F(n-2);
end;
```

(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];
```


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



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 F[i] := 1
        else
            F[i] := Lookup-F(i-1) +
                    Lookup-F(i-2);
        return F[i];
```

avoid recomputing

compute and save

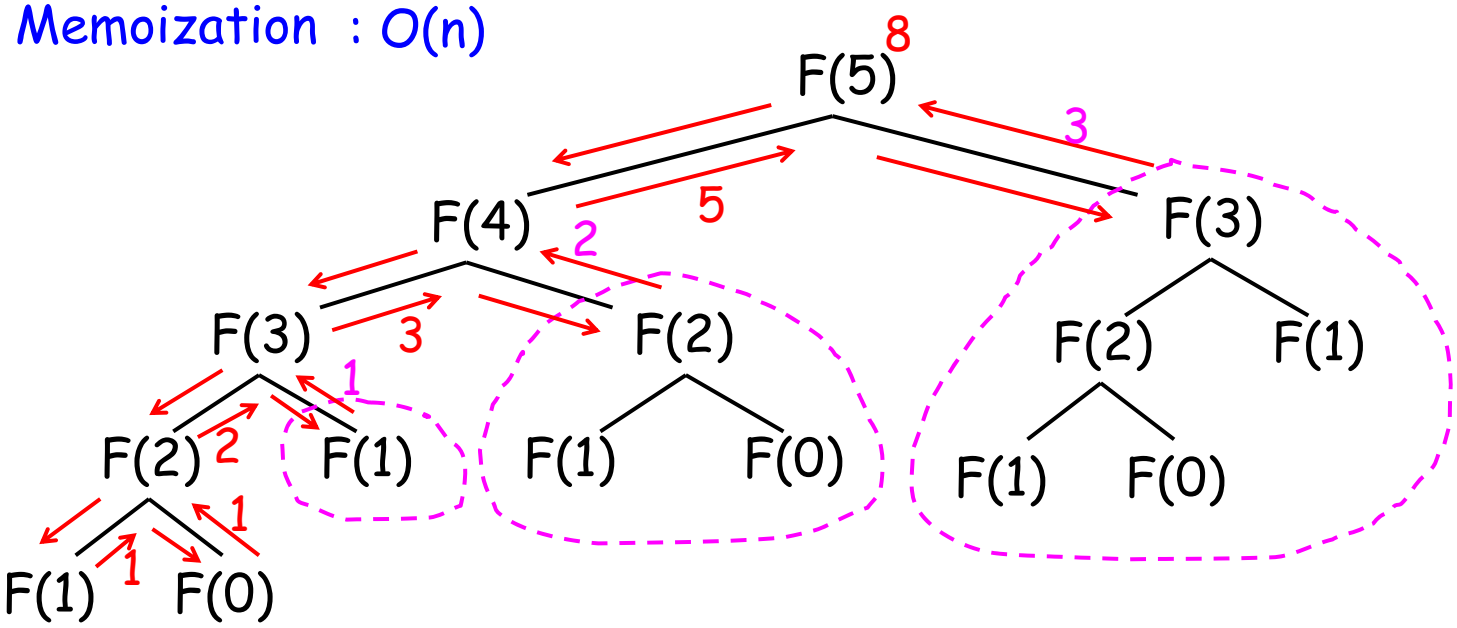
save for latter usage

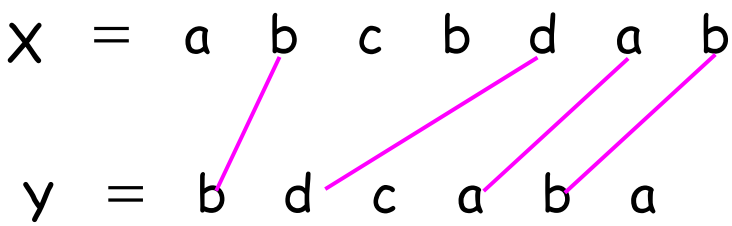
Top-down: ~~$O(2^n)$~~

\+

Memoization : $O(n)$

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

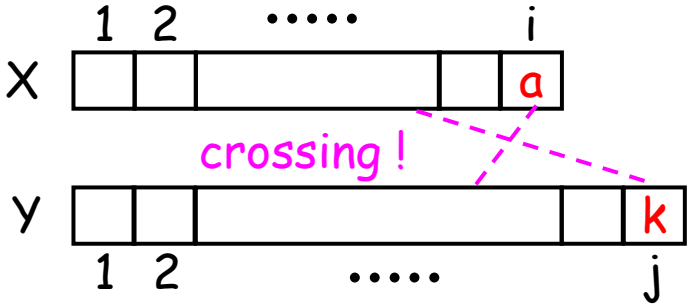




LCS ⇨ max # of non-crossing matching

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 !

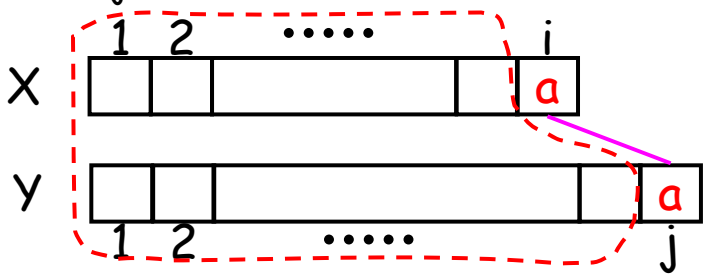
Let c[i,j] = LCS(X[1..i], Y[1..j])

discard y_j

$$c[i, j] = \text{MAX}\begin{cases} c[i, j-1] \\ c[i-1, j] \end{cases}$$

discard x_i

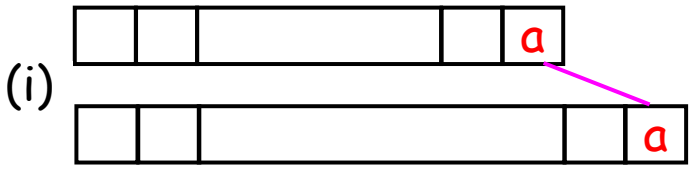
Case 1. $x_i = y_j$

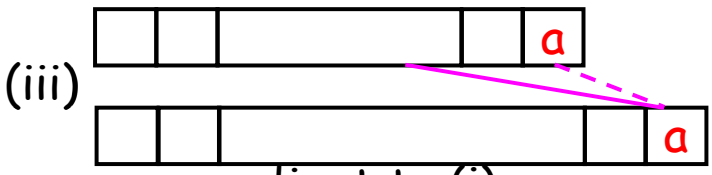


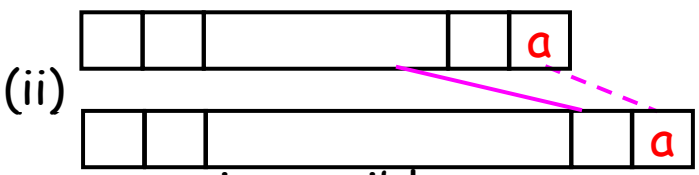
Match the tails !

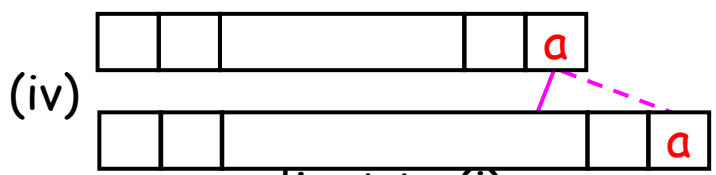
c[i, j] = c[i-1, j-1] + 1

---------- last matching of an optimal solution: 4 cases ---------

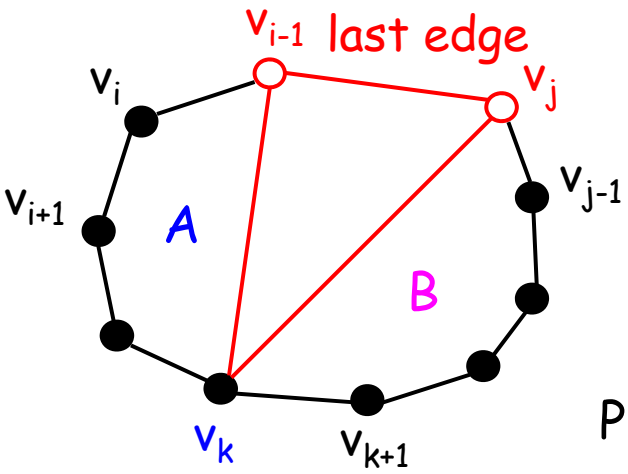








t[i,j] : optimal triangulation of P(v_{i-1}, v_i, …, v_j)

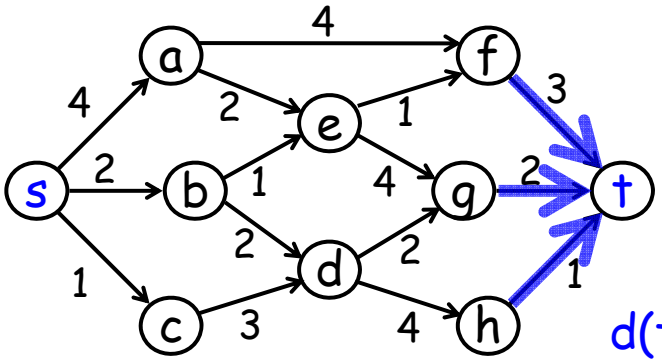


★ 包含 v_{i-1}

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

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

Directed Acyclic graph (multi-stage graph)



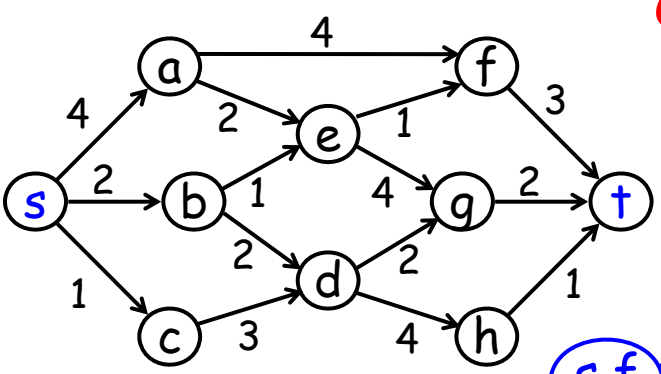
Given a DAG G = (V,E), find a shortest path from s to t

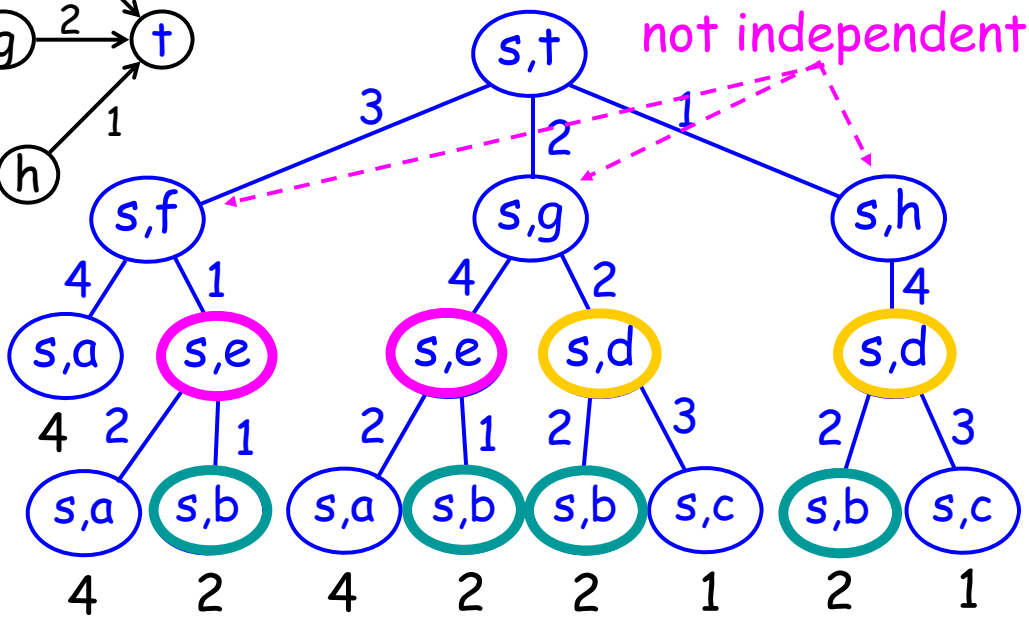
d(t) = MIN{ d(f)+3, d(g)+2, d(h)+1 }

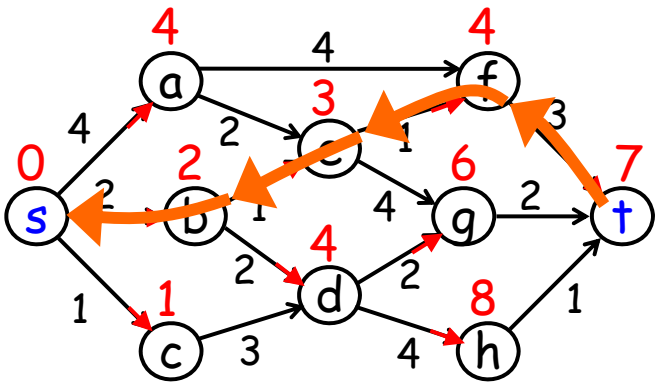
* d(v) : shortest distance from s to v

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

Overlapping Sub-problems







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

* 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

A simple exercise

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

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

n = 1 ⇨ {1}

n = 2 ⇨ {1, 1}

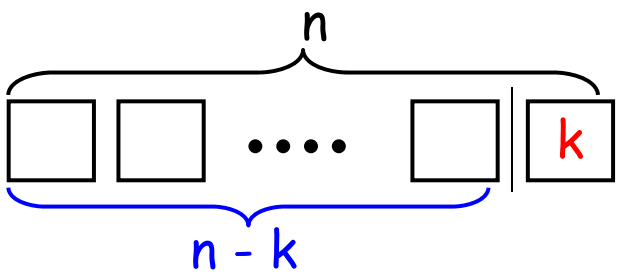
n = 3 ⇨ {3}

n = 4 ⇨ {1, 3}

n = 9 ⇨ {1, 3, 5}

optimal substructure

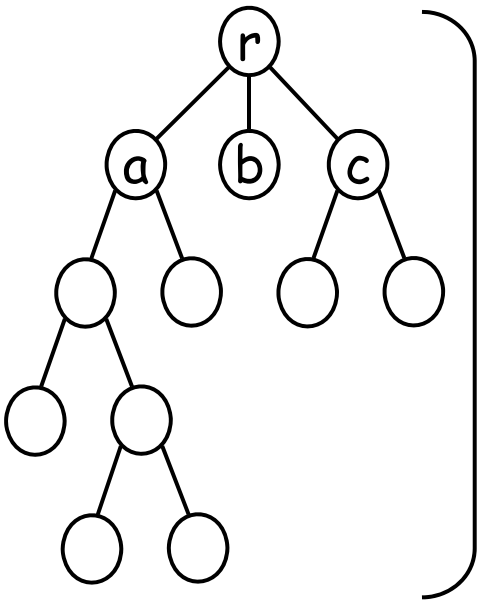
➡ F(9) = F(4) + 1



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

* F(0) = 0

Dynamic Programming ?



h=?

* 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