

Longest Common Subsequence

Longest Common Subsequence

Subsequence ของ string S คือเซตของอักขระที่ปรากฏในลำดับ จากซ้ายไปขวา ไม่จำเป็นต้องมาจากลำดับติดกัน

ตัวอย่างเช่น กำหนด string S เป็น ACTTGCC

- ตัวอย่าง subsequence ของ S

ACT, ATTC, T, AC, ACTTGC ทั้งหมดนี้เป็น subsequence

- ตัวอย่างที่ไม่ใช่ subsequence ของ S

TTA, ATGA

Common subsequence

Common subsequence ของ 2 string คือ subsequence ที่ปรากฏในทั้งสอง string

Longest common subsequence คือ common subsequence ที่มีความยาว(จำนวนตัวอักขระ) มากที่สุด

ตัวอย่าง

S1 = AAACCGTGAGTTATTCGTTCTAGAA

S2 = CACCCCTAAGGTACCTTTGGTTC

ลองหา Longest common subsequence ของ S1 กับ S2

ตัวอย่าง Longest Common Subsequence

S1 = AAACCGTGAGTTATTCTAGAA

S2 = CACCCCTAAGGTACCTTTGGTTC

LCS = ACCTAGTACTTTG

Bruteforce algorithm

สมมติว่า มี sting $x[1...m]$ และ $y[1..n]$ ที่ต้องการหา LCS

เราจะตรวจสอบทุก ๆ subsequence ของ x ว่า เป็น subsequence ของ y หรือไม่

วิเคราะห์

- หากเรามี subsequence อยู่ 1 แบบ การจะเปรียบเทียบว่า ทั้ง 2 string มี subsequence ตรงกันไหมใช้เวลาเท่าไร
 - ใช้เวลา $O(m+n)$
- กรณี string ยาว n ตัว จะสามารถสร้าง substring ได้ 2^n แบบ

ดังนั้น worst case running time ในการเปรียบเทียบทุกรูปแบบ คือ $O((m+n) 2^n)$

Dynamic Programming

1. นิยาม subproblem

พิจารณาความยาวของ longest common subsequence

ใช้ LCS หาค่าของมันเอง

เราจะแทนความยาวของ sequence s ด้วย $|s|$

แนวทาง เราจะพิจารณา prefixes ของ x และ y

นิยาม $c[i,j] = |\text{LCS}(x[1..i], y[1..j])|$

แล้ว $c[m,n] = |\text{LCS}(x,y)|$

Dynamic Programming

2. หา recurrence ของ subproblem

หากกำลังพิจารณา string x ถึงตำแหน่งอักขระที่ i

และพิจารณา string y ถึงตำแหน่งอักขระที่ j

กำหนดให้ $Z = z_1 z_2 \dots z_p$ เป็น LCS ที่เป็นคำตอบที่ดีที่สุดของ $x[1..j]$ และ $y[1..j]$ จะแบ่งได้ 2 กรณี

- 1) $x[i] = y[j]$ หมายถึงกรณีอักขระใน string x ตัวที่ i และอักขระใน string y ตัวที่ j เหมือนกัน
- 2) $x[i] \neq y[j]$ หมายถึงกรณีอักขระใน string x ตัวที่ i และอักขระใน string y ตัวที่ j ไม่เหมือนกัน

Dynamic Programming : หา recurrence

(1) กรณี $x[i] = y[j]$ หมายถึงกรณีอักขระใน string x ตัวที่ i และ
อักขระใน string y ตัวที่ j **เหมือนกัน**

เช่น กำหนด $x = \text{BDCABA}$ และ $y = \text{ABCBADAB}$

หากกำลังพิจารณาตำแหน่งที่ $i=5$ และ $j=4$ นั่นคือ $x[5] = y[4] = 'B'$

จะพบว่า $z_1 z_2 \dots z_{p-1}$ เป็นคำตอบที่ดีที่สุดของ $x[1..i-1]$ (“**BDCA**”)
และ $y[1..j-1]$ (“**ABC**”) นั่นคือมี “**BC**” เป็น LCS

ดังนั้น คำตอบที่ดีที่สุดของ $x[1..i]$ (“**BDCA****B**”) และ $y[1..j]$ (“**ABC****B**”)

จะเท่ากับ $1 +$ คำตอบที่ดีที่สุดของ $x[1..i-1]$ และ $y[1..j-1]$
นั่นคือ LCS เป็น “**BCB**”

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

Dynamic Programming : หา recurrence

(2) กรณีที่สอง $x[i] \neq y[j]$ หมายถึงกรณีอักขระใน string x ตัวที่ i และอักขระใน string y ตัวที่ j ไม่เหมือนกัน

เช่น กำหนด $x = \text{BDCABA}$ และ $y = \text{ABCBDAB}$

หากกำลังพิจารณาตำแหน่งที่ $i=6$ และ $j=4$ นั่นคือ $x[6] \neq y[4]$

พบว่าความยาวของคำตอบที่ดีที่สุดไม่เพิ่มขึ้น

ดังนั้น $z_1 z_2 \dots z_p$ จะเป็นคำตอบที่ดีที่สุดที่เป็นค่ามากกว่าระหว่าง

1) $x[1..i-1]$ (string “**BD**CAB”) และ $y[1..j]$ (string “A**BC**B”)

2) หรือ $x[1..i]$ (string “**BD**CABA”) และ $y[1..j-1]$ (string “A**BC**”)

นั่นคือ $c[i,j] = \max(c[i,j-1], c[i-1,j])$

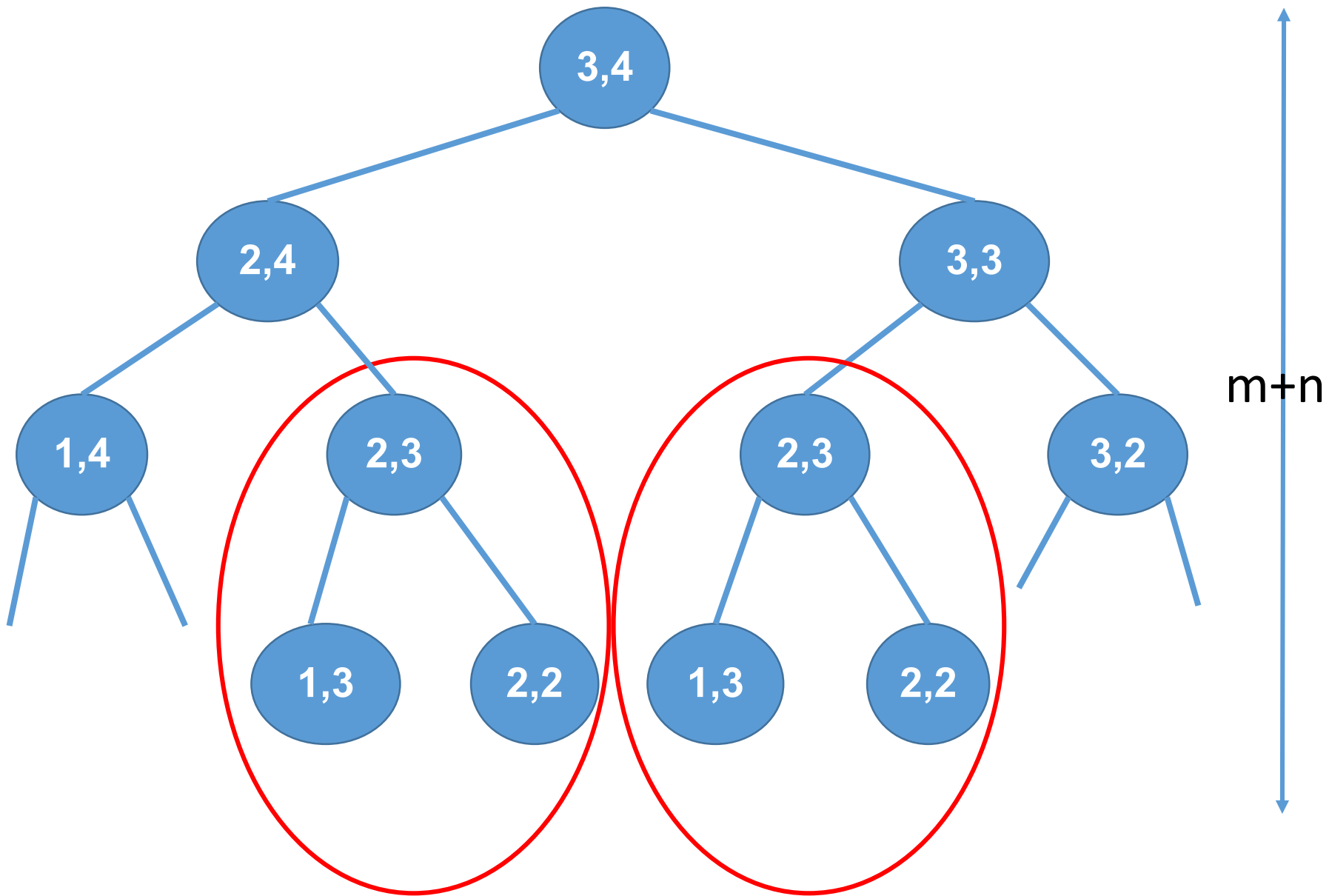
Dynamic Programming : หา recurrence

- เขียนสมการ recurrence ได้ดังนี้

$$c[i, j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j] \\ \max\{c[i-1, j], c[i, j-1]\} & \text{กรณีอื่นๆ} \end{cases}$$

หากเขียน algorithm แบบ recursive จะได้ว่า

```
LCS(x, y, i, j){
    if x[i]==y[j] then
        c[i, j] = LCS(x, y, i-1, j-1)+1
    else c[i, j] = max{LCS(x, y, i-1, j), LCS(x, y, i, j-1)}
}
```



Dynamic programming

3.หา Base case

ถ้า string y ไม่มีสัักตัว (ยาว=0 ตัว) LCS ควรเป็นเท่าไร

for i=1 to m

$$c[i,0] = 0$$

ถ้า string x ไม่มีสัักตัว (ยาว=0 ตัว) LCS ควรเป็นเท่าไร

for j=1 to n

$$c[0,j] = 0$$

Dynamic programming : Algorithm

LCS(x,y)

for(i=0 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($x[i]=y[j]$)

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

 else $c[i,j] = \max\{c[i-1,j], c[i,j-1]\}$

Running Time= $O(mn)$ เนื่องจากคำนวณและเรียกใช้แต่ละช่อง
เสียเวลาเป็น constant

Space = $O(mn)$

Dynamic programming : Example

- X=BDCABA

เริ่มต้น

- Y=ABCB DAB

		A	B	C	B	D	A	B
B								
D								
C								
A								
B								
A								

Dynamic programming : Example

- X=BDCABA
- Y=ABCB DAB

ใส่ค่า Base case

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0							
D	0							
C	0							
A	0							
B	0							
A	0							

Dynamic programming : Example

- X=BDCABA
- Y=ABCB DAB

$x[1] \neq y[1]$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0						
D	0							
C	0							
A	0							
B	0							
A	0							

Dynamic programming : Example

- X=BDCABA
- Y=ABCBDAB

$x[1]=y[2]$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1					
D	0							
C	0							
A	0							
B	0							
A	0							

Dynamic programming : Example

- X=BDCABA
- Y=ABCBDAB

$x[1] \neq y[3]$


		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1 → 1					
D	0							
C	0							
A	0							
B	0							
A	0							

Dynamic programming : Example

- X=BDCABA
- Y=ABCB DAB

$x[1]=y[4]$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1			
D	0							
C	0							
A	0							
B	0							
A	0							



Dynamic programming : Example

- X=BDCABA
- Y=ABCB DAB

$x[1] \neq y[5]$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1		
D	0							
C	0							
A	0							
B	0							
A	0							

Dynamic programming : Example

- X=BDCABA
- Y=ABCBDAB

$x[1] \neq y[6]$


		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	
D	0							
C	0							
A	0							
B	0							
A	0							

Dynamic programming : Example

$x[1]=y[7]$

- X=BDCABA
- Y=ABCBDAB

		A	B	C	B	D	A	B
B	0	0	0	0	0	0	0	0
D	0	0	1	1	1	1	1	1
C	0							
A	0							
B	0							
A	0							



Dynamic programming : Example

- X=BDCABA
- Y=ABCB DAB

เทียบกับ $x[2]$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0							
A	0							
B	0							
A	0							

Fill จนวนครบ

- X=BDCABA
- Y=ABCBDAB

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Fill จวนครบ

- X=BDCABA
- Y=ABCBBDAB

สร้างตารางย้อนกลับ

เหมือนกัน ให้ใส่ ↖

หากเลือกจากด้านบน ให้ใส่ ↑

หากเลือกจากด้านซ้าย ให้ใส่ ←

จุดที่เป็น ↖

คือคำตอบ

	A	B	C	B	D	A	B
	0	0	0	0	0	0	0
B	0	← 0	↖ 1	← 1	↖ 1	← 1	↖ 1
D	0	← 0	↑ 1	← 1	↖ 1	↖ 2	← 2
C	0	← 0	↑ 1	↖ 2	← 2	← 2	← 2
A	0	↖ 1	← 1	↑ 2	← 2	↖ 3	← 3
B	0	↑ 1	↖ 2	← 2	↖ 3	← 3	↖ 4
A	0	↖ 1	↑ 2	← 2	↑ 3	↖ 4	← 4

Fill จวนครบ

- X=BDCABA
- Y=ABCB DAB

สร้างตารางย้อนกลับ

เหมือนกัน ให้ใส่ ↖

หากเลือกจากด้านบน ให้ใส่ ↑

หากเลือกจากด้านซ้าย ให้ใส่ ←

จุดที่เป็น ↖

คือคำตอบ

		A	B	C	(B)	(D)	(A)	(B)
(B)	0	0	0	0	0	0	0	0
(D)	0	0	1	1	1	1	1	1
C	0	0	1	1	1	2	2	2
(A)	0	1	1	2	2	2	3	3
(B)	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Homework

- จงเขียนโปรแกรมสำหรับแก้ปัญหา LCS โดยให้ผลลัพธ์ของโปรแกรมแสดงคำตอบ LSC และขนาดLCS ที่ได้
 - เป็นงานกลุ่ม กลุ่มละไม่เกิน 3 คน
 - ให้ส่งงานโปรแกรมผ่าน email: benjamas.p@cmu.ac.th
 - ส่งโปรแกรมภายในวันจันทร์ที่ 3 พ.ย. 2557

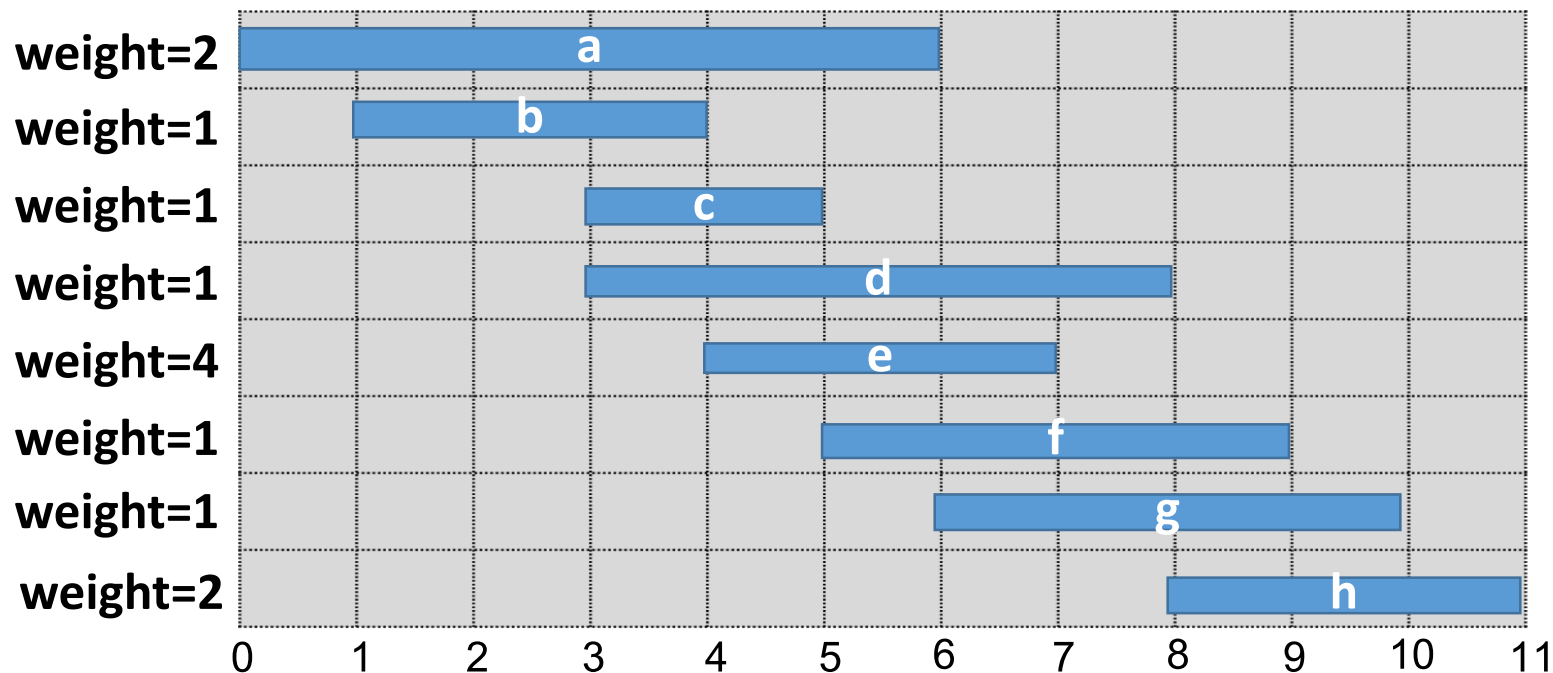
Weighted Interval Scheduling

Weighted Interval Scheduling

- กำหนดให้ งาน j เริ่มเวลา s_j เลิกเวลา f_j และมีน้ำหนักหรือค่า v_j
งานสองงานใดๆ จะสอดคล้องกันถ้าไม่ใช่เวลาร่วมกัน

Goal: หาเซตของงานที่สอดคล้องกันที่มีค่าน้ำหนักรวมมากที่สุด

คำถาม: จะหา algorithm ในการแก้ปัญหานี้อย่างไร



Greedy algorithm

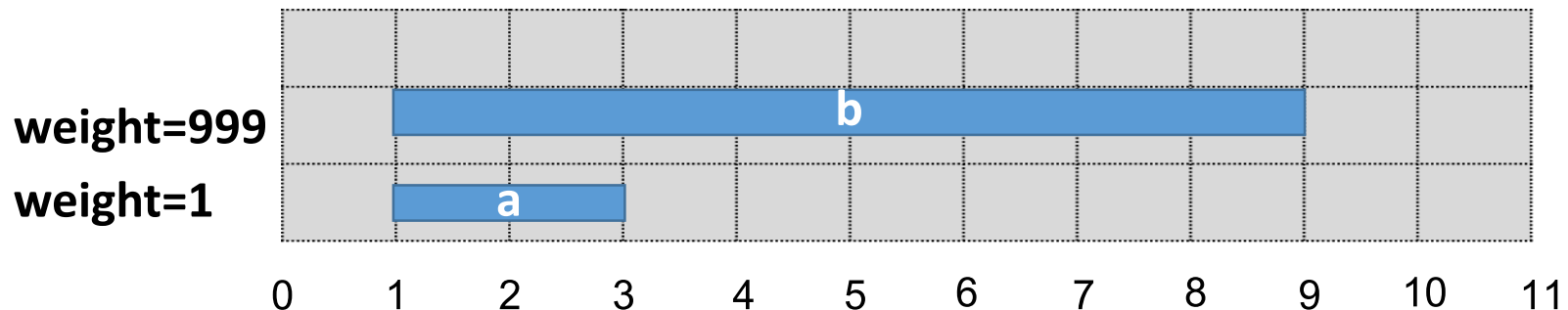
ทบทวน Unweighted Interval Scheduling

Greedy algorithm จะใช้ได้กับกรณีที่ weight เป็น 1

- พิจารณางานโดยเรียงตามลำดับเวลาเสร็จจากน้อยไปมาก
- เพิ่มงานไปยังเซตคำตอบถ้าสอดคล้องกับงานที่ถูกเลือกไว้ก่อนหน้านี้

คำถาม: จะเกิดอะไรขึ้นถ้าเรานำเอา greedy algorithm ของ Interval scheduling มาใช้กับ weighted interval scheduling

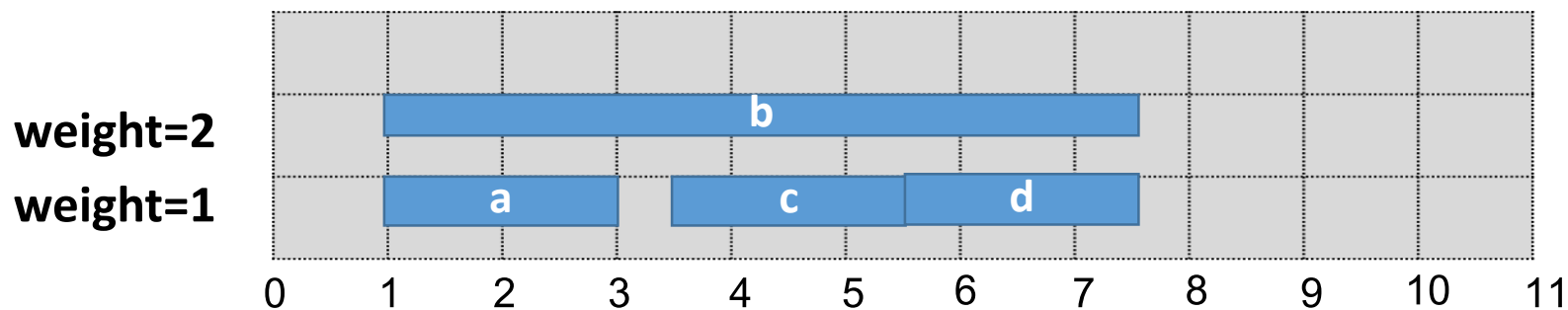
คำตอบ: มีข้อขัดแย้ง



Greedy algorithm

คำถาม: จะเกิดอะไรขึ้นถ้าเราใช้วิธี greedy โดยใส่งานที่มีน้ำหนักมากที่สุดก่อน

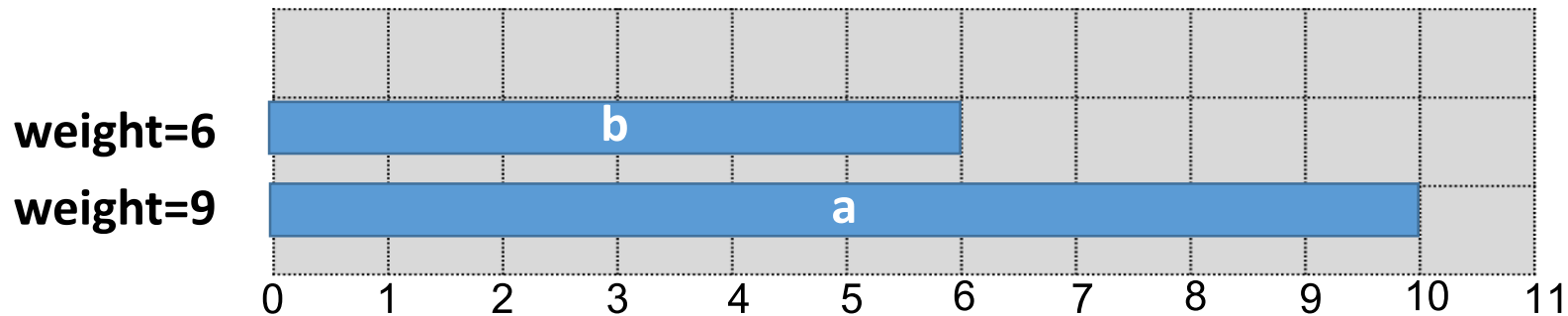
คำตอบ: ก็ยังมีข้อขัดแย้ง



Greedy algorithm

คำถาม: จะเกิดอะไรขึ้นถ้าเราใช้วิธี greedy โดยใช้เวลา
น้ำหนักต่อเวลา

คำตอบ: ก็ยังมีข้อขัดแย้ง



Brute-force algorithm

คำถาม: เราอาจจะต้องลองทุกแบบที่เป็นไปได้ แล้วเราจะทำอย่างไร

คำตอบ: ใช้ backtracking การย้อนกลับ

คำถาม: จำนวนของรูปแบบการเลือกงานที่เป็นไปได้ทั้งหมดไม่เกินเท่าไร (n^2 , n^3 , 2^n , $2!$)

คำตอบ: กรณีแย่สุด $O(2^n)$ แต่บางรูปแบบอาจจะเกิดขึ้นไม่ได้

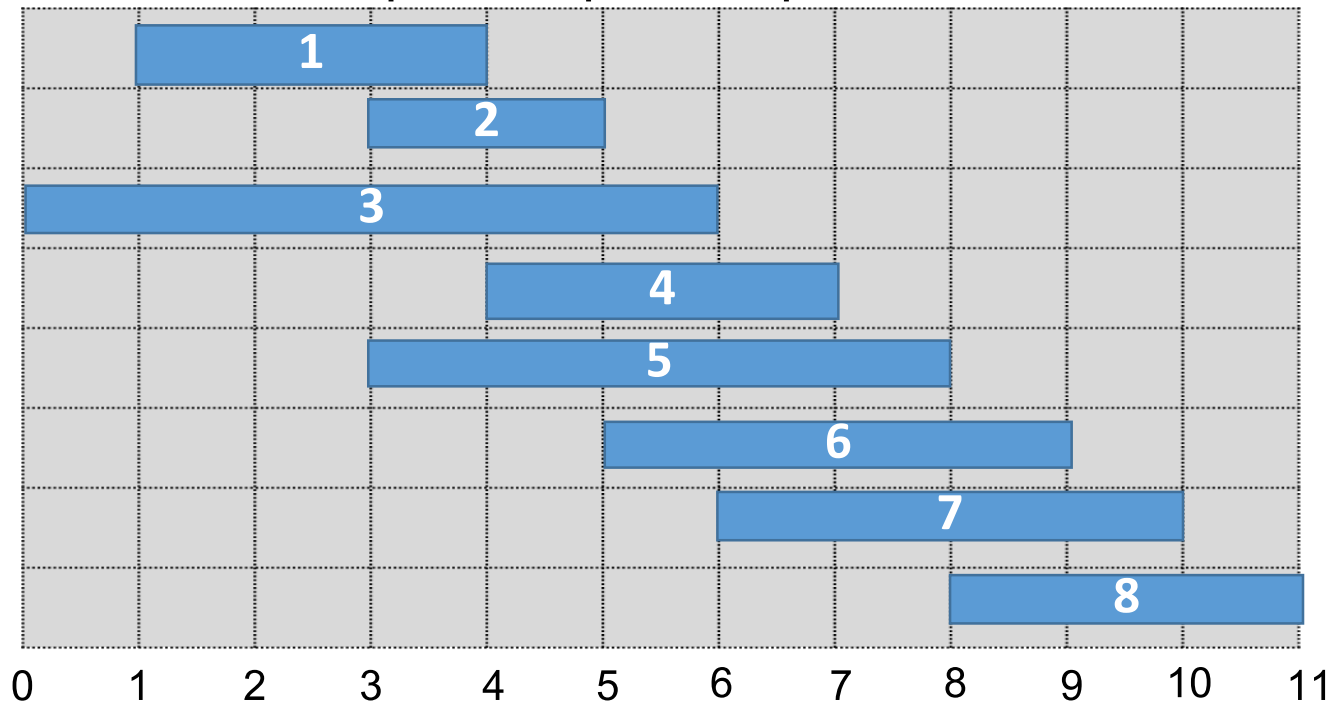
- แล้วเราจะทำให้ดีขึ้นได้อย่างไร

Dynamic programming

- เราจะกำหนดป้ายชื่อให้กับงานใหม่ตามเวลาเสร็จงาน $f_1 \leq f_2 \leq \dots \leq f_n$

นิยาม $p(j)$ แทน index i ที่มากที่สุดที่ $i < j$ และ i สอดคล้องกับ j

ตัวอย่างเช่น $p(8)=5$, $p(7)=3$, $p(2)=0$



j	p(j)
0	-
1	0
2	0
3	0
4	1
5	0
6	2
7	3
8	5

Dynamic programming

1. นิยาม subproblem

ให้ $S(j)$ แทนค่าของคำตอบที่ดีที่สุดที่ประกอบด้วยงานที่ $1, 2, \dots, j$

เราพบว่ามี 2 กรณี

- กรณีแรก j ถูกเลือก

ไม่สามารถใช้งานที่ไม่สอดคล้อง $\{p(j), p(j+1), \dots, j-1\}$

จะต้องรวมเข้ากับคำตอบที่ดีที่สุดของปัญหาที่ประกอบด้วยงานที่ $1, 2, \dots, p(j)$

- กรณีที่สอง j ไม่ถูกเลือก

ดังนั้นจะมีค่าเท่ากับคำตอบที่ดีที่สุดของปัญหาที่ประกอบด้วยงานที่ $1, 2, \dots, j-1$

$$S[j] = \max\{v_j + S[p(j)], S[p(j-1)]\}$$

Dynamic programming

2. หา Base case

หากไม่มีสั้กงานตอบอะไร

$$S[0] = 0$$

3. หาสมการ Recurrence ของ problem

$$S[j] = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + S[p(j)], S[j - 1]\} & \text{กรณีอื่นๆ} \end{cases}$$

Improve Complexity by using recursive algorithm

Input: $n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n, v_1, v_2, \dots, v_n$

Sort jobs by finish time so that $f_1 \leq f_2, \dots, < f_n$

Compute $p(1), p(2), \dots, p(n)$

Compute-opt(j){

 if(j=0)

 return 0

 else

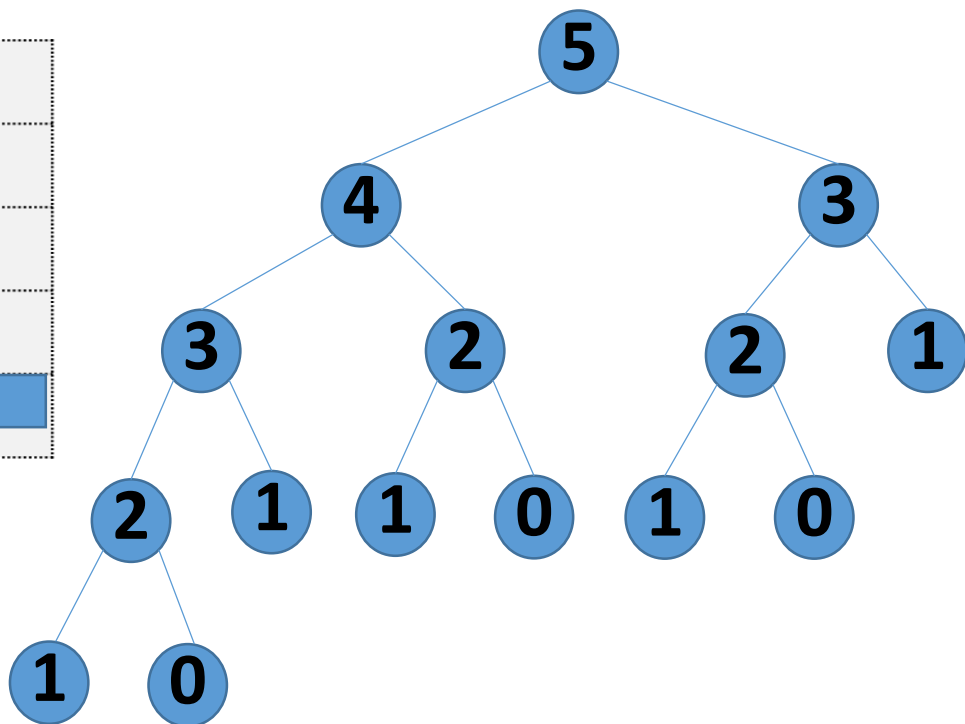
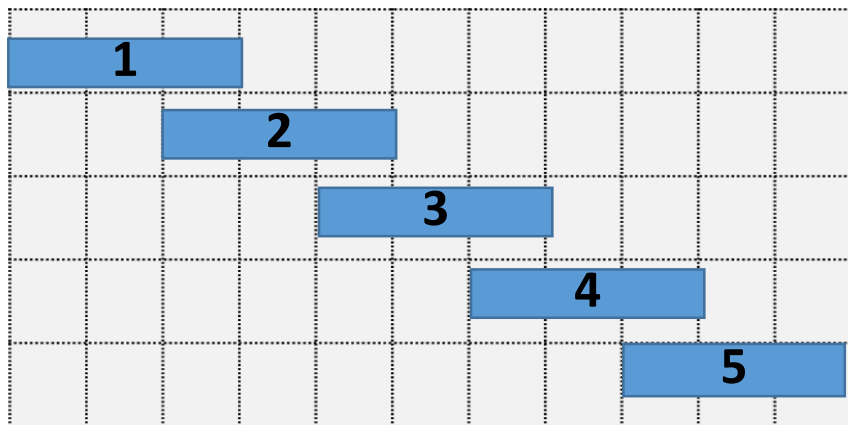
 return $\max(v_j + \text{Compute-opt}(p(j)), \text{Compute-opt}(j-1))$

}

Brute-force algorithm

จากการสังเกตเราพบว่าเมื่อใช้ recursive อย่างเดียวจะพบว่ามี
ปัญหาย่อยที่ซ้ำกันมาก => exponential algorithm

ตัวอย่างเช่น จำนวนของการเวียน recursive จะมีลักษณะโตคล้าย
กับ Fibonacci sequence



Improve Complexity using Iterative Computation

Input: $n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n, v_1, v_2, \dots, v_n$

Sort jobs by finish time so that $f_1 \leq f_2, \dots, < f_n$

Compute $p(1), p(2), \dots, p(n)$

Iterative-Compute-opt(j){

$S[0]=0$

for($j=1$ to n)

$s[j] = \max(v_j + s[p(j)], S[j-1])$

}

return $S[n]$

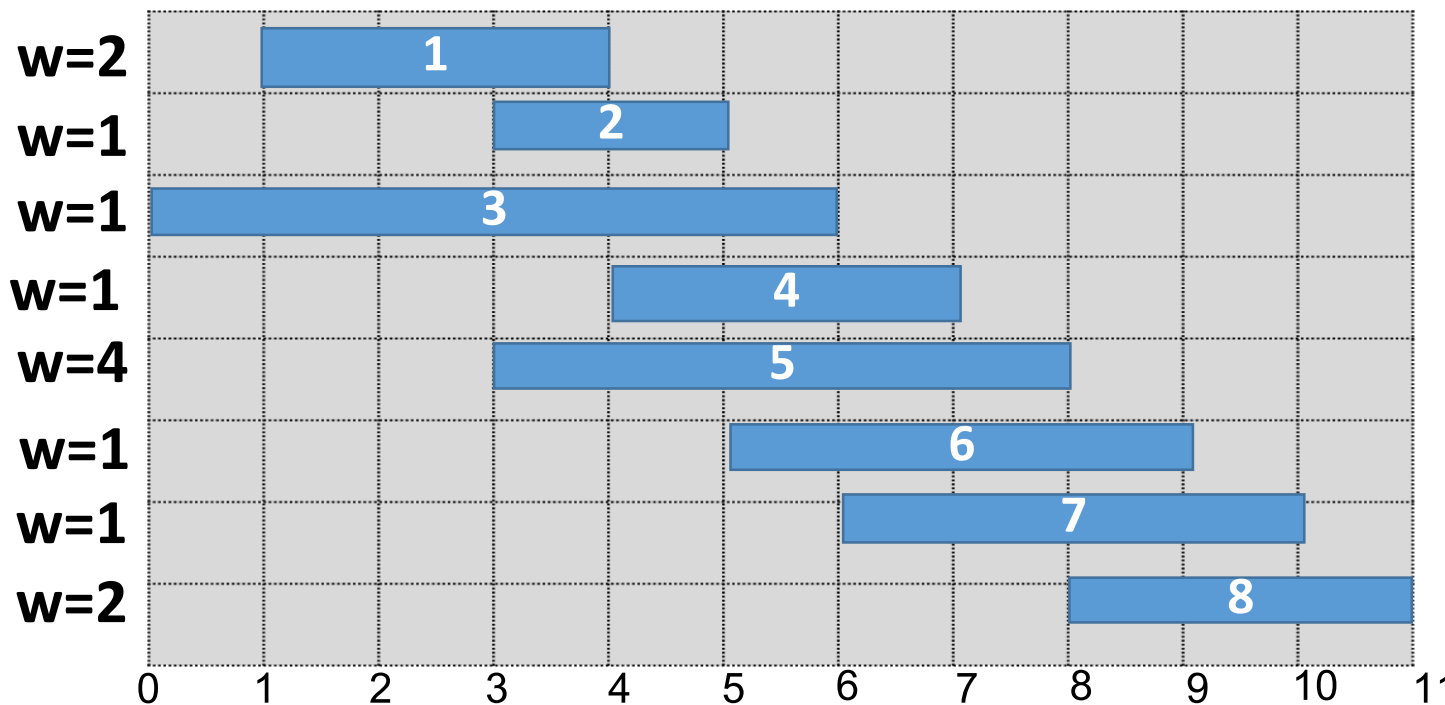
$S[j]$ = ค่าของคำตอบที่ดีที่สุดสำหรับงาน 1 ถึง j

Weighted Interval Scheduling

เราจะกำหนดป้ายชื่อให้กับงานใหม่ตามเวลาเสร็จงาน $f_1 \leq f_2 \leq \dots \leq f_n$

นิยาม $p(j)$ แทน index i ที่มากที่สุดที่ $i < j$ และ i สอดคล้องกับ j

ตัวอย่างเช่น $p(8)=5$, $p(7)=3$, $p(2)=0$



j	p(j)	s(j)
0	-	0
1	0	
2	0	
3	0	
4	1	
5	0	
6	2	
7	3	
8	5	