Weighted Interval Scheduling

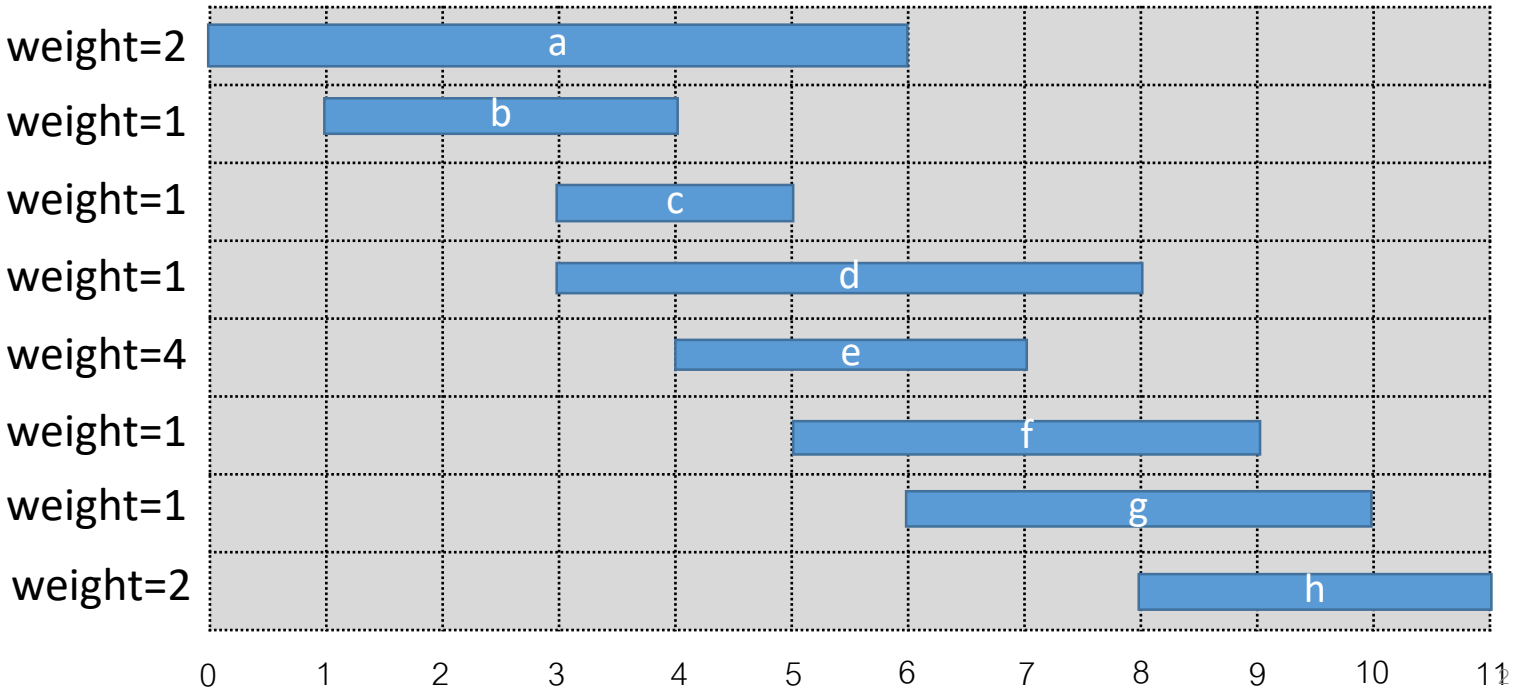
Weighted Interval Scheduling

กำหนดให้ งาน j เริ่มเวลา s_j เลิกเวลา f_j และมีน้ำหนักหรือค่า v_j

งานสองงานใดๆ จะสอดคล้องกันถ้าไม่ใช้เวลาร่วมกัน

Goal: หาซับเซตของงานที่สอดคล้องกันที่มีค่าน้ำหนักรวมมากที่สุด

คำถาม: จะหา algorithm ในการแก้ปัญหานี้อย่างไร



ทบทวน Unweighted Interval Scheduling

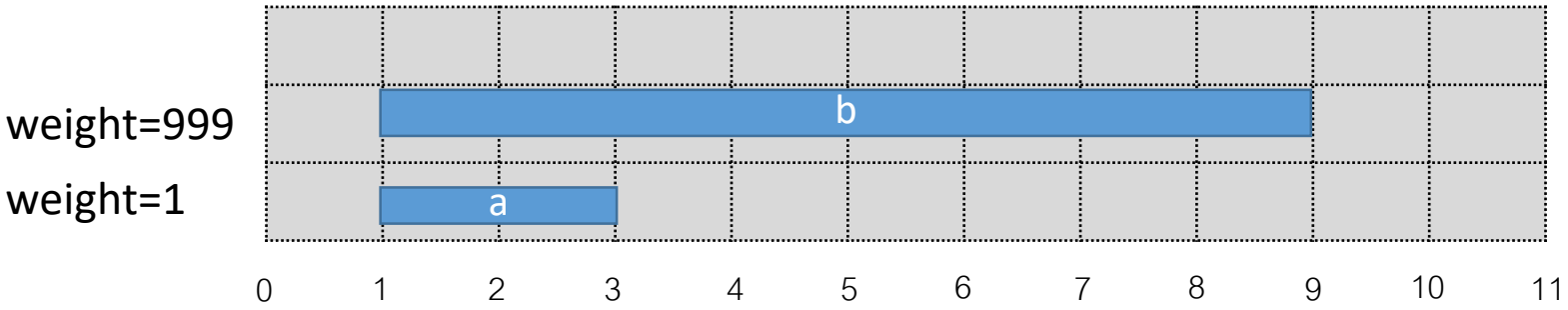
ทบทวน: Greedy algorithm นั้นทำงานได้กับกรณีที่ weight เป็น 1

พิจารณางานโดยเรียงตามลำดับเวลาเสร็จจากน้อยไปมาก

เพิ่มงานไปยังเซตคำตอบถ้าสอดคล้องกับงานที่ถูกเลือกไว้ก่อนหน้า

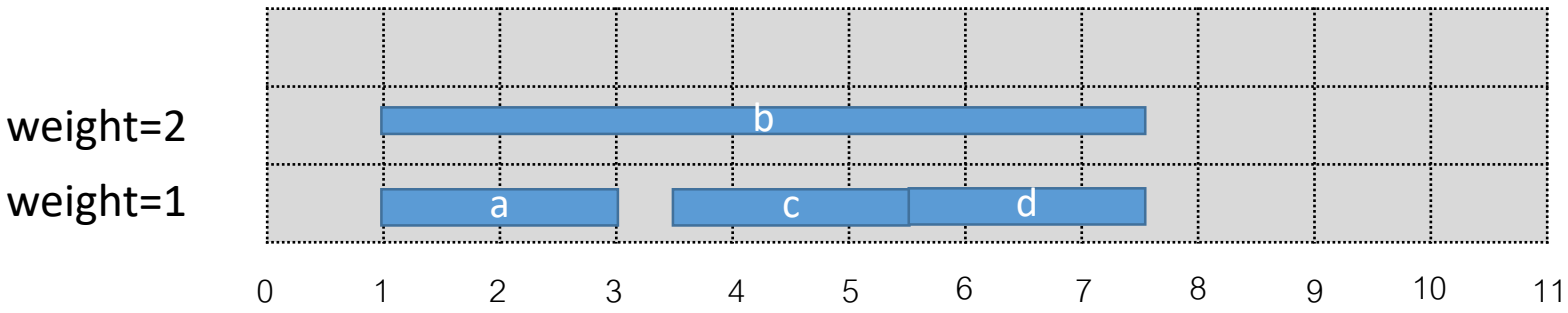
คำถาม: จะเกิดอะไรขึ้นถ้าเรานำเอา greedy algorithm ของ Interval scheduling มาใช้กับ weighted interval scheduling

คำตอบ: มีข้อขัดแย้ง



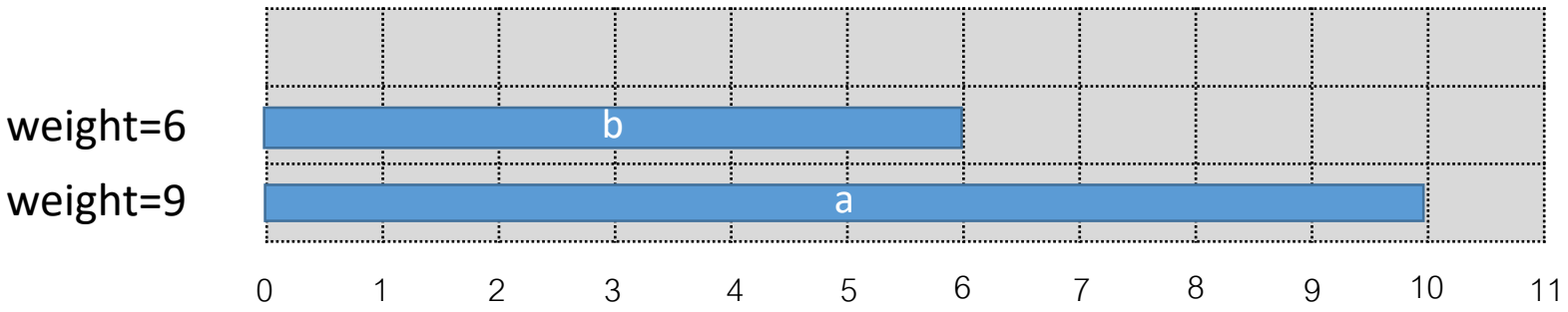
คำถาม: จะเกิดอะไรขึ้นถ้าเราใช้วิธี greedy โดยใส่งานที่มีน้ำหนักมากสุดก่อน

คำตอบ: ก็ยังมีข้อขัดแย้ง



คำถาม: จะเกิดอะไรขึ้นถ้าเราใช้วิธี greedy โดยใช้การเรียงน้ำหนักต่อเวลา

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

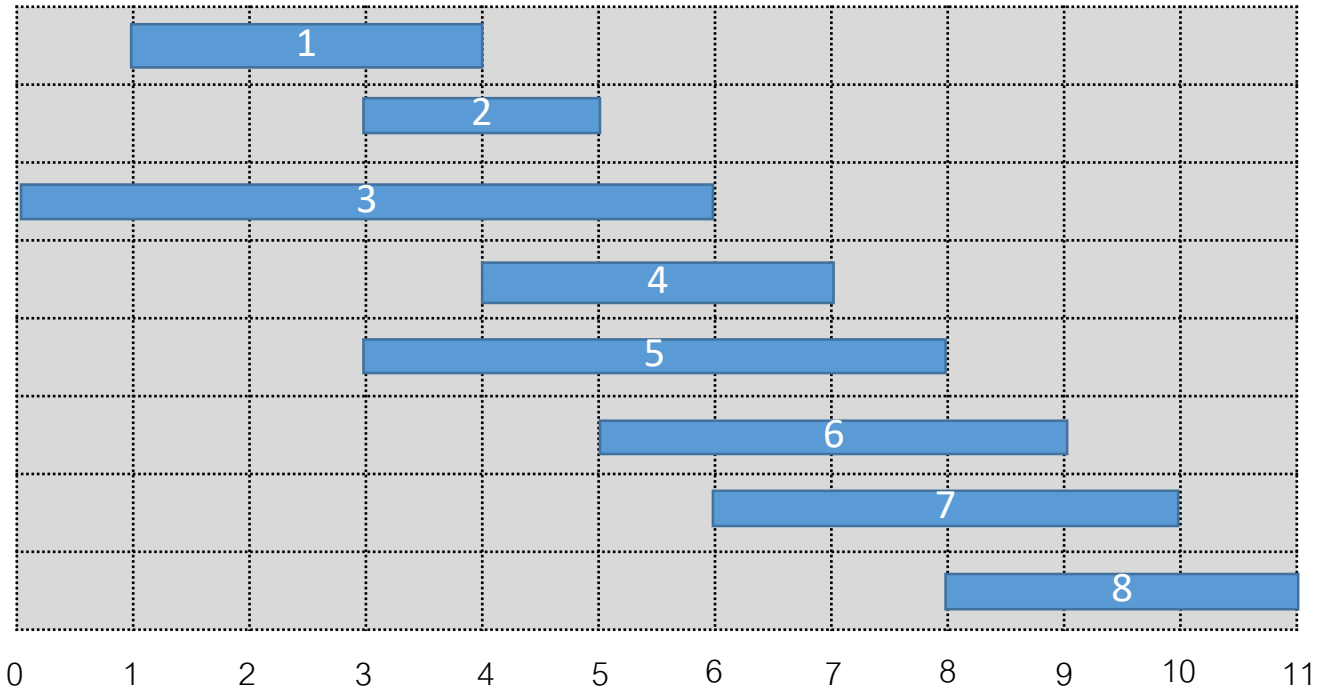
- แล้วเราจะทำให้ดีขึ้นได้อย่างไร

Weighted Interval Scheduling

เราจะกำหนดป้ายชื่อให้กับงานใหม่ตามเวลาเสร็จงาน $f_1 \leq f_2 \leq \ldots \leq f_n$

นิยาม p(j) แทน index i ที่มากสุดที่ 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

นิยาม subproblem

ให้ S(j) แทนค่าของคำตอบที่ดีที่สุดที่ประกอบด้วยงานที่ 1, 2, …, j

เราพบว่ามี 2 กรณี

กรณีแรก j ถูกเลือก

ไม่สามารถใช้งานที่ไม่สอดคล้อง {p(j), p(j+1), … j-1}

จะต้องรวมเข้ากับคำตอบที่ดีที่สุดของปัญหาที่ประกอบด้วยงานที่ 1, 2, …, p(j)

กรณีที่สอง j ไม่ถูกเลือก

ดังนั้นจะมีค่าเท่ากับคำตอบที่ดีที่สุดของปัญหาที่ประกอบด้วยงานที่ 1, 2, …, j-1

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

Base case

หากไม่มีสักงานตอบอะไร

$S[0] = 0$

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

Brute force algorithm

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

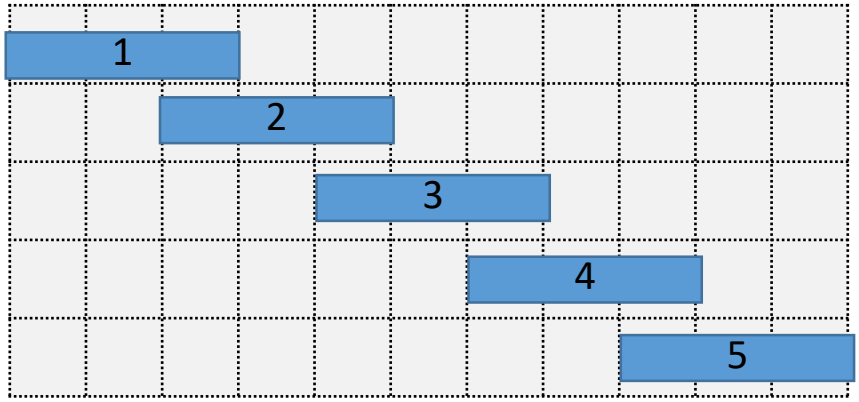
Sort jobs by finish time so that $f_1 \leq f_2, \ldots, <f_n$

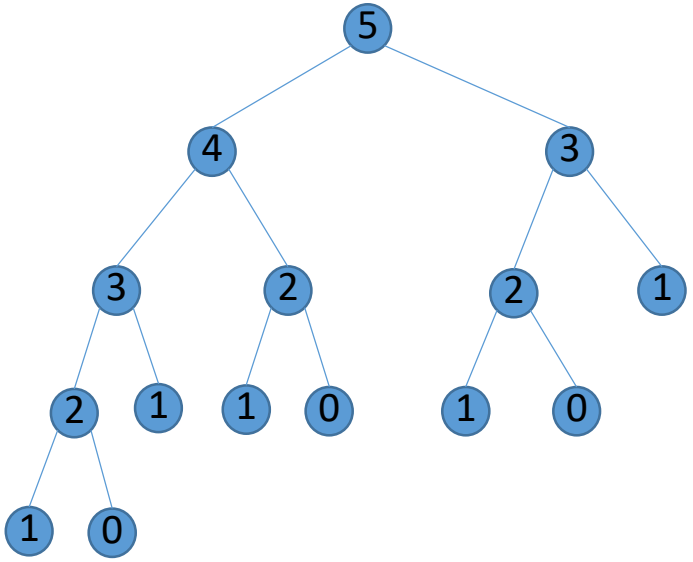
Compute p(1), p(2),…,p(n)

```
Compute-opt(j){
    if(j=0)
        return 0
    else
        return max(v_j+Compute-opt(p(j)),Compute-opt(j-1)
}
```

จากการสังเกตเราพบว่าเมื่อใช้ recursive อย่างเดียวจะพบว่ามีปัญหาย่อยที่ซ้อนกันมาก => exponential algorithm

ตัวอย่างเช่น จำนวนของการเวียน recursive จะมีลักษณะโตคล้ายกับ Fibonacci sequence





p(1)=0, p(j)=j-2

Improve Complexity

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

Sort jobs by finish time so that $f_1 \leq f_2, \ldots, <f_n$ O(nlgn)

Compute p(1), p(2),…,p(n) $O(n^2)$->O(nlgn)

```
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]
```

O(n)

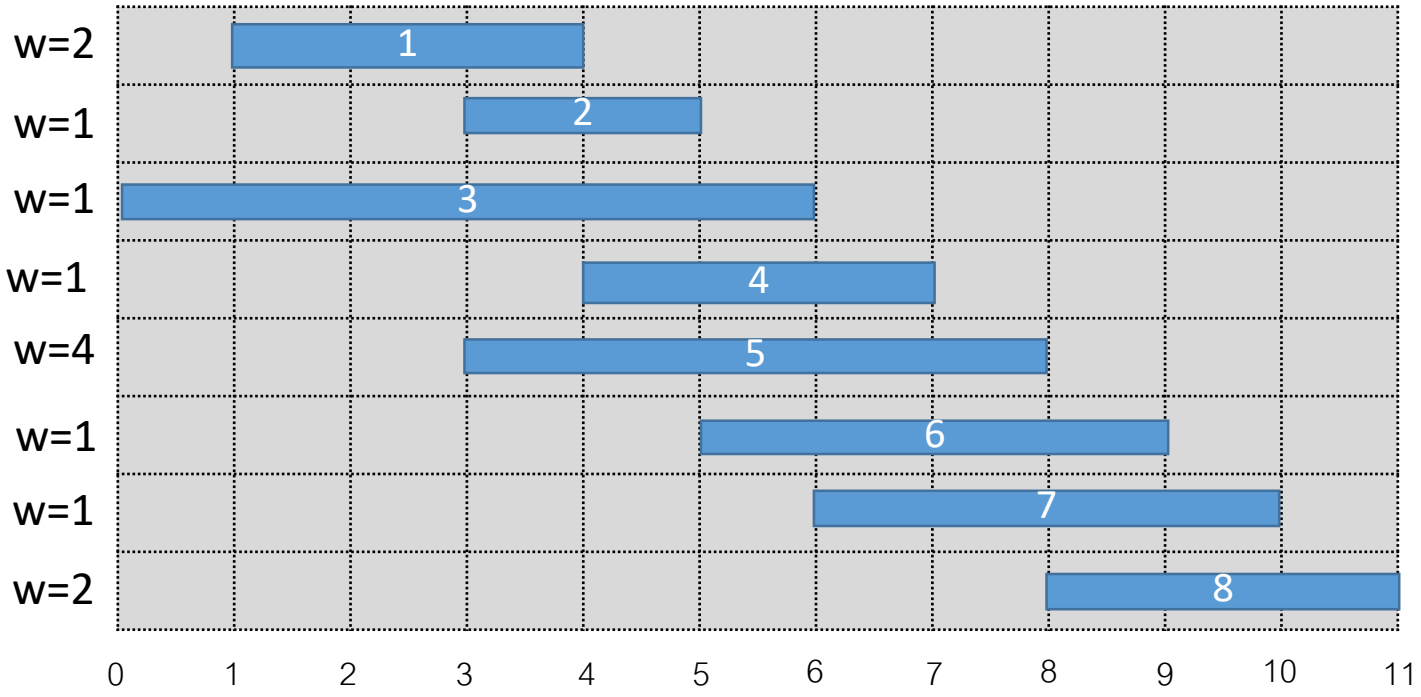
S[j] = ค่าของคำตอบที่ดีที่สุดสำหรับงาน 1 ถึง j

Weighted Interval Scheduling

เราจะกำหนดป้ายชื่อให้กับงานใหม่ตามเวลาเสร็จงาน $f_1 \leq f_2 \leq \ldots \leq f_n$

นิยาม p(j) แทน index i ที่มากสุดที่ 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	