

Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281

ผู้สอน: ตอน 1 อ. เบญจมาศ ปัญญางาม เรียน ห้อง 201

ตอน 2 อ. ดร. จักริน ขวชาติ เรียน ห้อง 209

)

บทที่ 9

Dynamic Programming Part IV

Longest Common Subsequence

Longest Common Subsequence

Subsequence ของ string S คือเซตของอักขระที่ปรากฏในลำดับจากซ้ายไปขวา 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(จำนวนตัวอักษร) มากที่สุด

ตัวอย่าง

S1 = AAACCGTGAGTTATTCGTTCTAGAA

S2 = CACCCCTAAGGTACCTTTGGTTC

ลองหา Longest common subsequence ของ S1 กับ S2

อ. ดร. จักริน ขวชาติ

อ. เบญจมาศ ปัญญางาม

ตัวอย่าง Longest Common Subsequence

S1 = AAACCGTGAGTTATTCGTTCTAGAA

S2 = CACCCTAAGGTACCTTTGGTTC

LCS = ACCTAGTACTTTG

Bruteforce algorithm

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

เราจะตรวจสอบทุกๆ subsequence ของ x ว่า เป็น subsequence ของ y หรือไม่

วิเคราะห์

- หากเรามี subsequence อยู่ 1 แบบ การจะเปรียบเทียบว่าทั้ง 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 ในการเปรียบเทียบทุกรูปแบบคือ $O((m+n) 2^n)$

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 **ไม่** เหมือนกัน

Dynamic Programming : หา recurrence

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

เช่น กำหนด $x=BDCA$ **BA** และ $y=ABC$ **BDAB**

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**”) นั่นคือ มี “**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

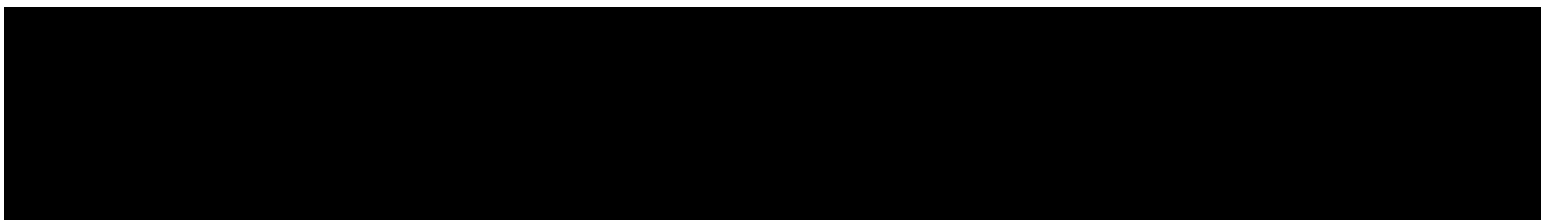
เช่น กำหนด $x=\text{BDCABA}$ และ $y=\text{ABCBDA}$

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 “BDCAB”) และ $y[1..j]$ (string “ABCB”)
 - 2) หรือ $x[1..i]$ (string “BDCABA”) และ $y[1..j-1]$ (string “ABC”)
- นั่นคือ $c[i,j] = \max(c[i, j-1], c[i-1, j])$

Dynamic Programming : หา recurrence

- เขียนสมการ recurrence ได้ดังนี้



หากเขียน algorithm แบบ recursive จะได้ว่า

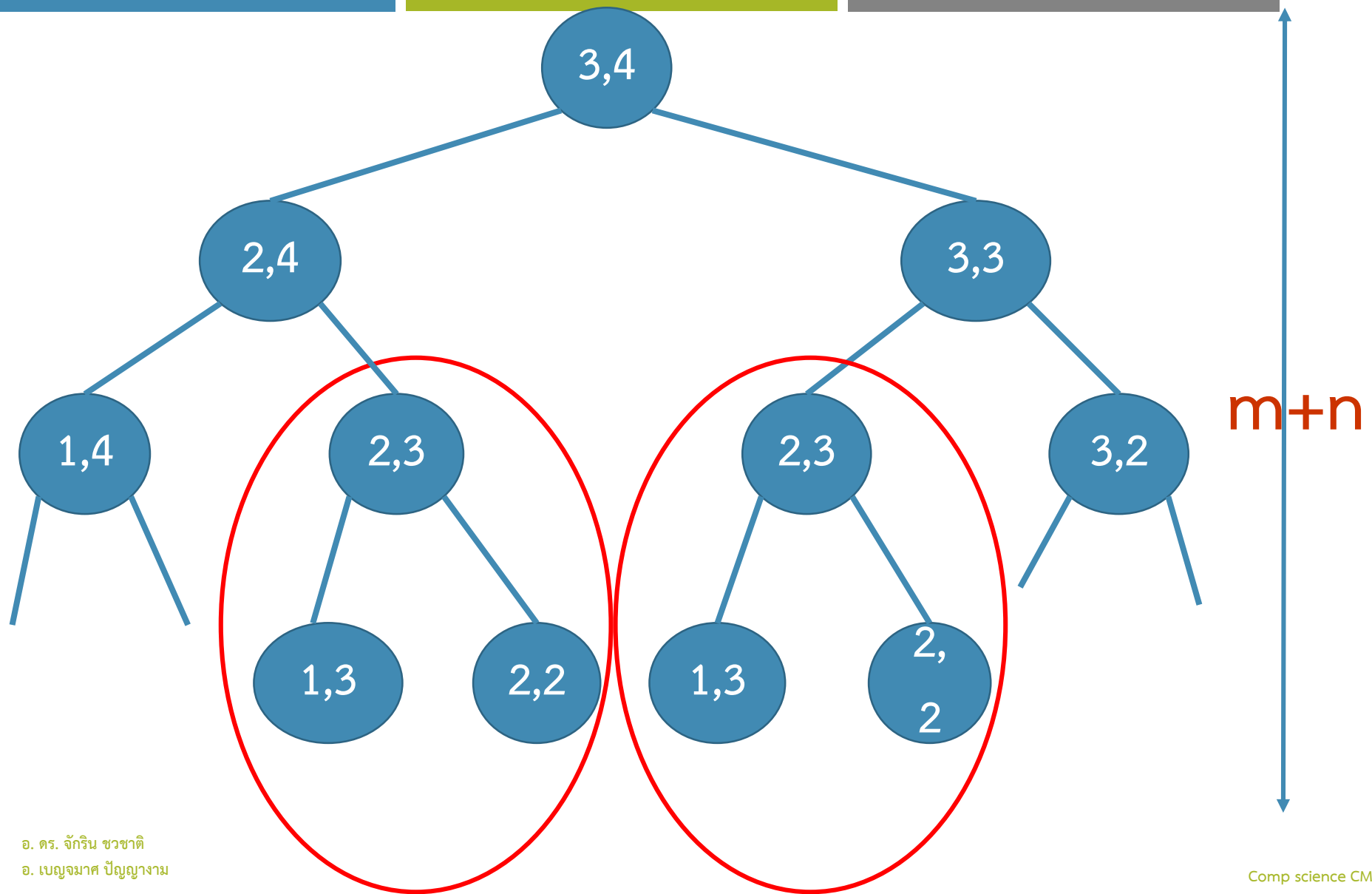
LCS(x, y, i, j){

if $x[i] == y[j]$ then

$c[i, j] = \text{LCS}(x, y, i-1, j-1) + 1$

else $c[i, j] = \max\{\text{LCS}(x, y, i-1, j), \text{LCS}(x, y, i, j-1)\}$

}



Dynamic programming

3.หา Base case

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

for $i=1$ to m

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

ถ้า string x ไม่มีสักรตัว (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= $O(mn)$ เนื่องจากคำนวณและเรียกใช้แต่ละช่องเสียเวลาเป็น constant

Space = $O(mn)$

Dynamic programming : Example

เริ่มต้น

□ X=BDCABA

□ Y=ABCB DAB

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

Dynamic programming : Example

ใส่ค่า Base case

□ X=BDCABA

□ Y=ABCBDAB

		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

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

❑ Y=ABCBDA

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

Dynamic programming : Example

□ X=BDCABA

$x[1]=y[2]$

□ Y=ABCBDA

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

Dynamic programming : Example

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

❑ X=BDCABA

❑ Y=ABCBADB

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

Dynamic programming : Example

❑ X=BDCABA

$x[1]=y[4]$

❑ Y=ABCBDA

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

Dynamic programming : Example

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

❑ X=BDCABA

❑ Y=ABCBADB

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

Dynamic programming : Example

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

❑ X=BDCABA

❑ Y=ABCBADB

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


Dynamic programming : Example

❑ X=BDCABA

$x[1]=y[7]$

❑ Y=ABCBDA

		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

เทียบกับ x[2]

❑ Y=ABCB DAB

		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							
B	0							
A	0							

Fill จินตกรรม

❑ $X = \text{BDCABA}$

❑ $Y = \text{ABCBDAB}$

		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	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

Fill จวนครบ

สร้างตารางย้อนกลับ

เหมือนกัน ให้ใส่ ↖

หากเลือกจากด้านบน ให้ใส่ ↑

หากเลือกจากด้านซ้าย ให้ใส่ ←

จุดที่เป็น ↖ คือคำตอบ

❑ X=BDCABA

❑ Y=ABCBDAB

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

Fill จวนครบ

สร้างตารางย้อนกลับ

เหมือนกัน ให้ใส่

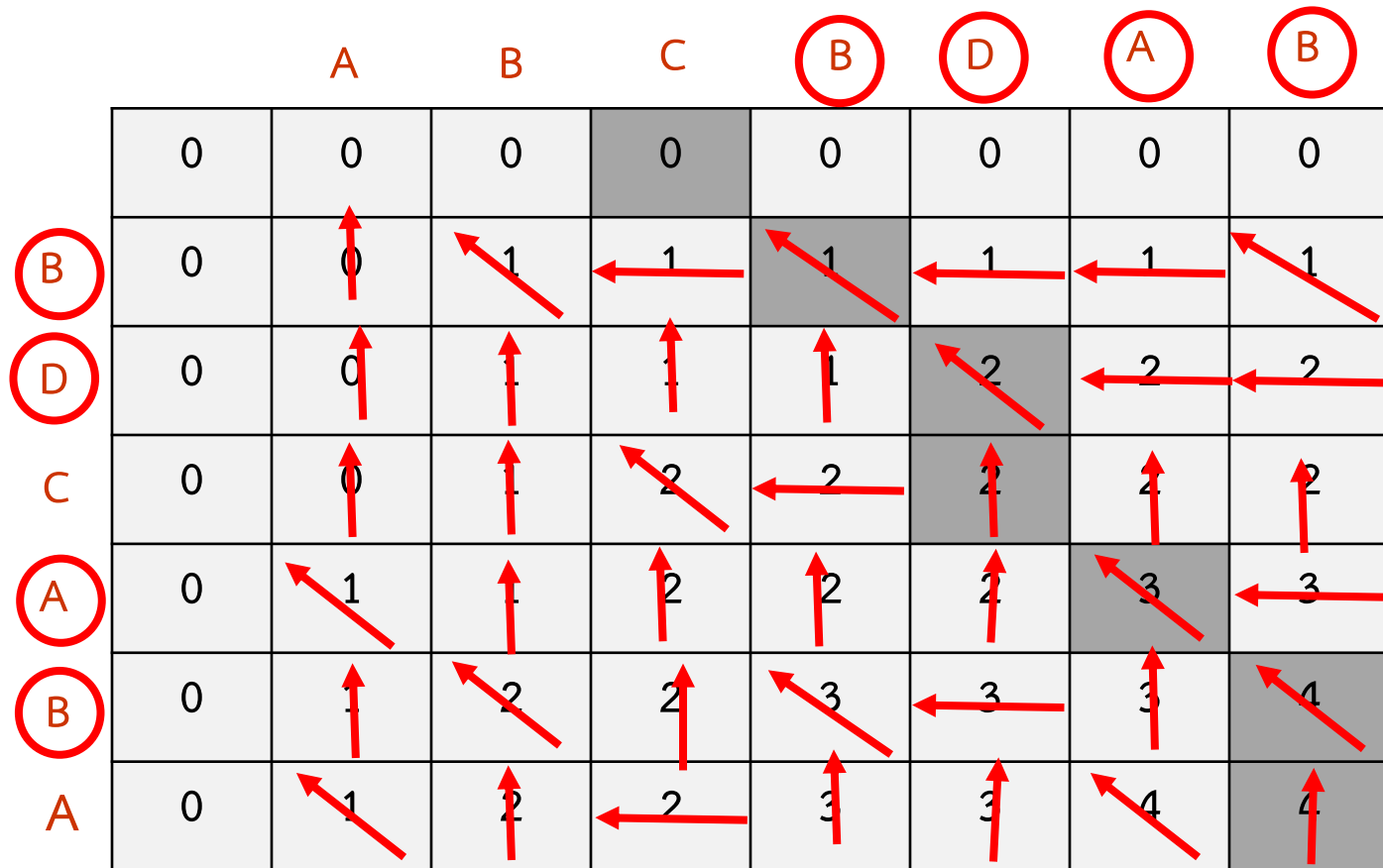
หากเลือกจากด้านบนให้ใส่

หากเลือกจากด้านซ้ายให้ใส่

จุดที่เป็น คือคำตอบ

❑ X=BDCABA

❑ Y=ABCBDAB



แบบฝึกหัด

- 1) จงเขียนตาราง $c[i, j]$ เพื่อค้นหาคำตอบที่เป็น LCS ระหว่างสตริง $X = \text{BACBAD}$ และสตริง $Y = \text{ABAZDC}$
 - 2) จงเขียนอัลกอริทึมในการค้นหาและพิมพ์ค่าสตริงที่เป็นคำตอบของปัญหา LCS ระหว่างสตริง X และสตริง Y จากตาราง $c[i, j]$ เมื่อกำหนดให้ m คือความยาวสตริง X และ n คือความยาวสตริง Y
- Grader

Weighted Interval Scheduling

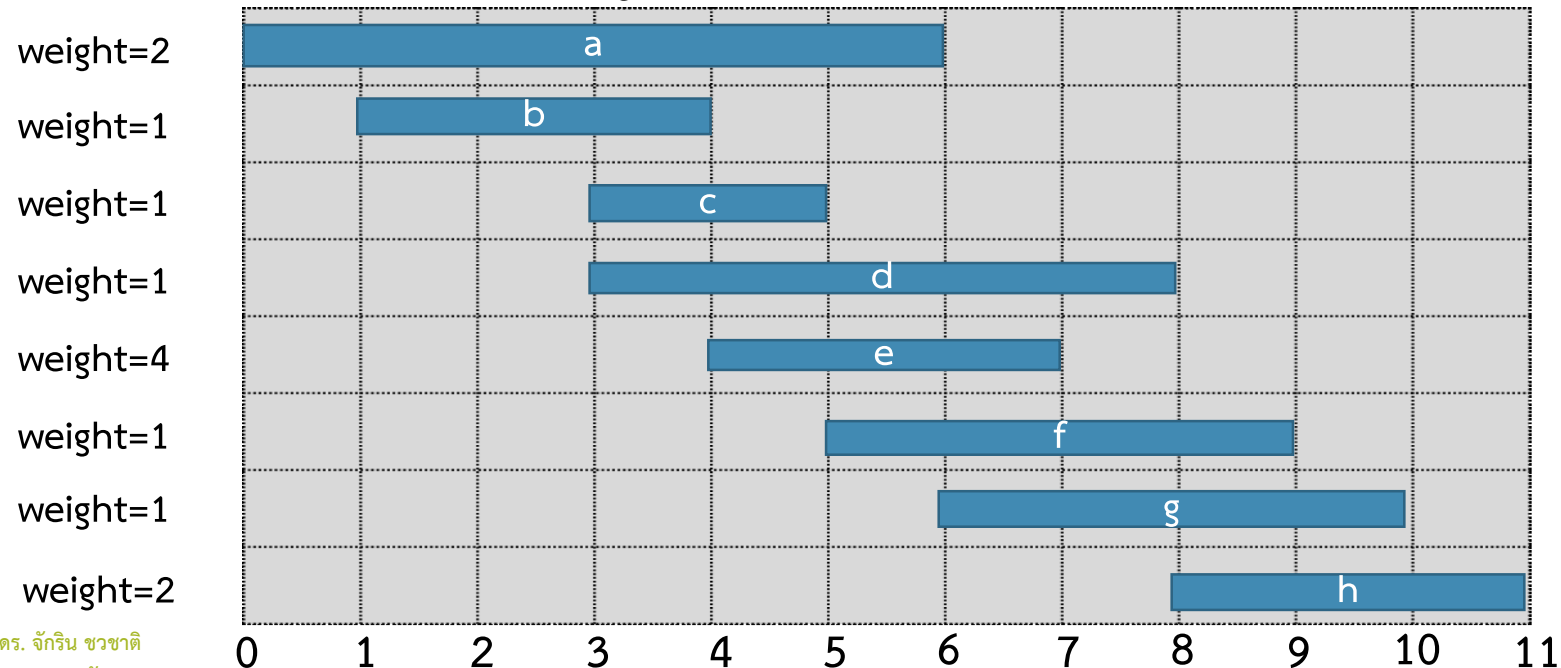
Weighted Interval Scheduling

กำหนดให้ งาน j เริ่มเวลา s_j เลิกเวลา f_j และมีน้ำหนักหรือค่า v_j

งานสองงานใดๆ จะสอดคล้องกันถ้าไม่ใช้เวลาร่วมกัน

Goal: หาซ้บเซตของงานที่สอดคล้องกันที่มีค่าน้ำหนักรวมมากที่สุด

คำถาม: จะหา algorithm ในการแก้ปัญหานี้อย่างไร



Greedy algorithm

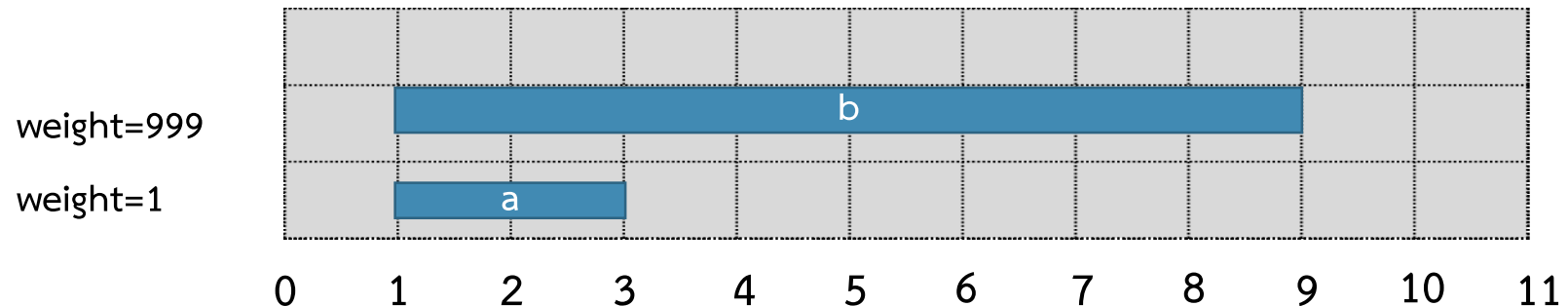
ทบทวน Unweighted Interval Scheduling

Greedy algorithm จะใช้ได้กับกรณีที่มี weight เป็น 1

- พิจารณางานโดยเรียงตามลำดับเวลาเสร็จจากน้อยไปมาก
- เพิ่มงานไปยังเซตคำตอบถ้าสอดคล้องกับงานที่ถูกละเลือกไว้ก่อนหน้านี้

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

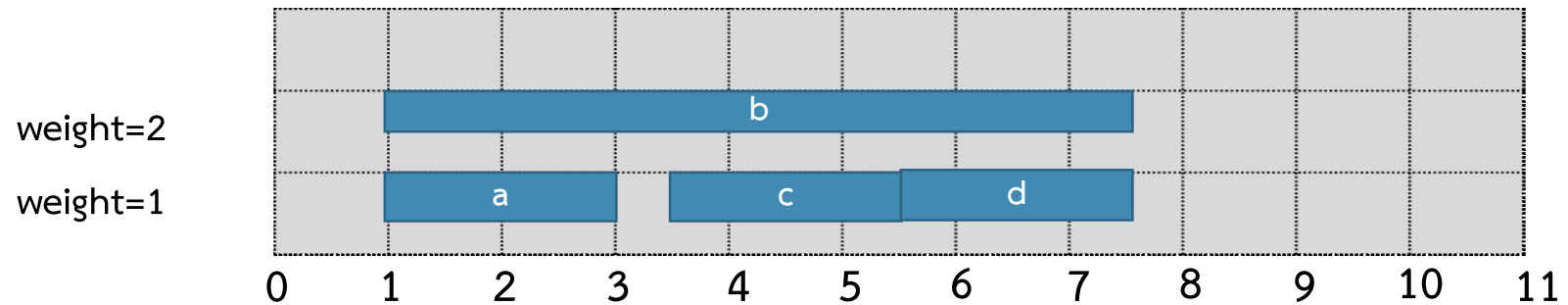
คำตอบ: มีข้อขัดแย้ง



Greedy algorithm

คำถาม: จะเกิดอะไรขึ้นถ้าเราใช้วิธี greedy โดยใส่งานที่มีน้ำหนักมากที่สุดก่อน

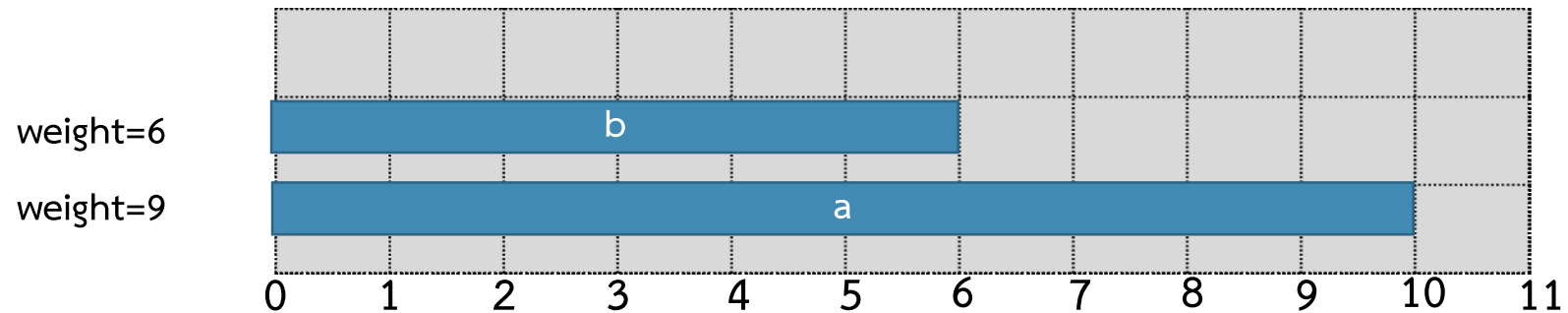
คำตอบ: ก็ยังมีข้อขัดแย้ง



Greedy algorithm

คำถาม: จะเกิดอะไรขึ้นถ้าเราใช้วิธี greedy 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

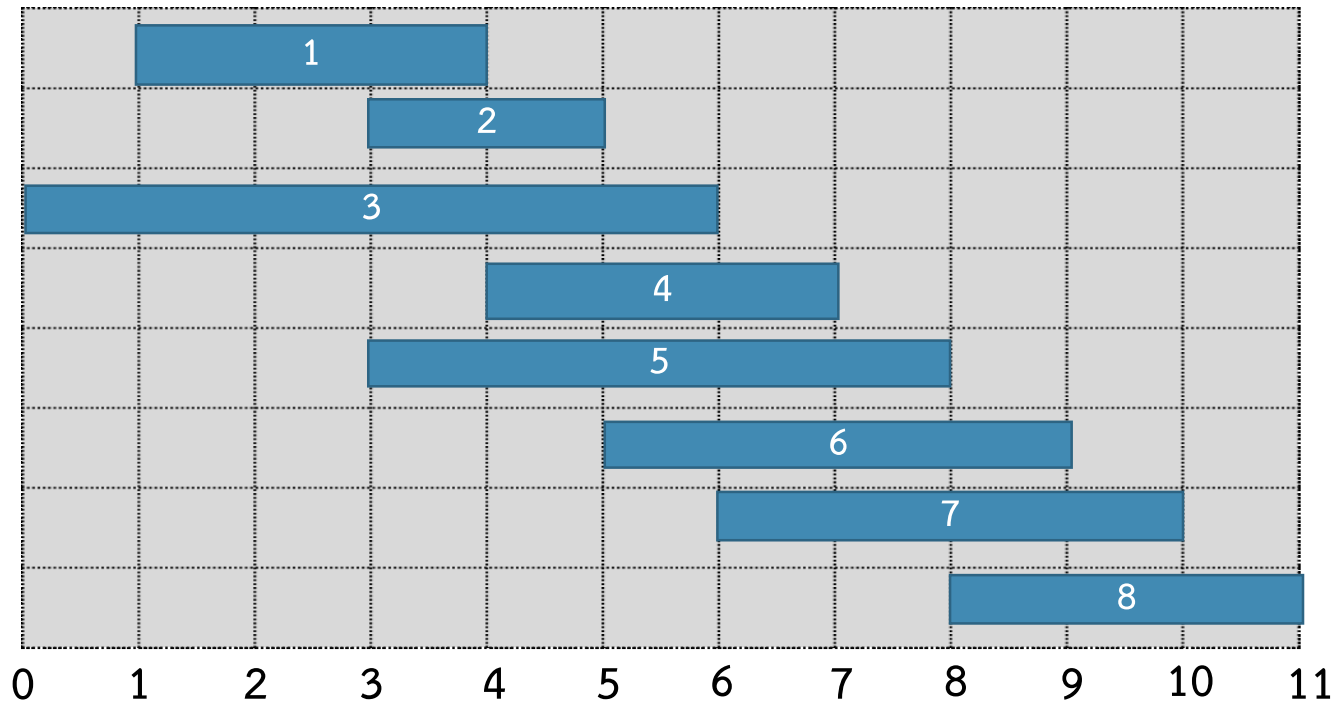
□ แล้วเราจะทำให้ดีขึ้นได้อย่างไร

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, ..., j

เราพบว่ามี 2 กรณี

□ กรณีแรก j ถูกเลือก

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

จะต้องรวมเข้ากับคำตอบที่ดีที่สุดของปัญหาที่ประกอบด้วยงานที่

1, 2, ..., $p(j)$

□ กรณีที่สอง j ไม่ถูกเลือก

ดังนั้นจะมีค่าเท่ากับคำตอบที่ดีที่สุดของปัญหาที่ประกอบด้วยงานที่ 1, 2, ..., $j-1$

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

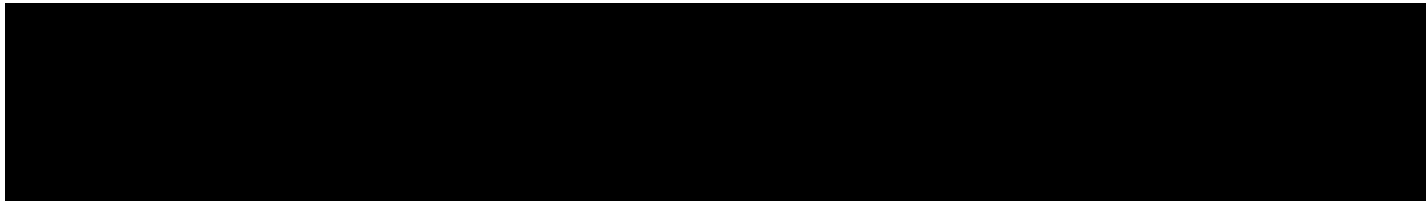
Dynamic programming

2. หา Base case

หากไม่มีสักรงานตอบอะไร

$$S[0] = 0$$

3. หาสมการ Recurrence ของ problem



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

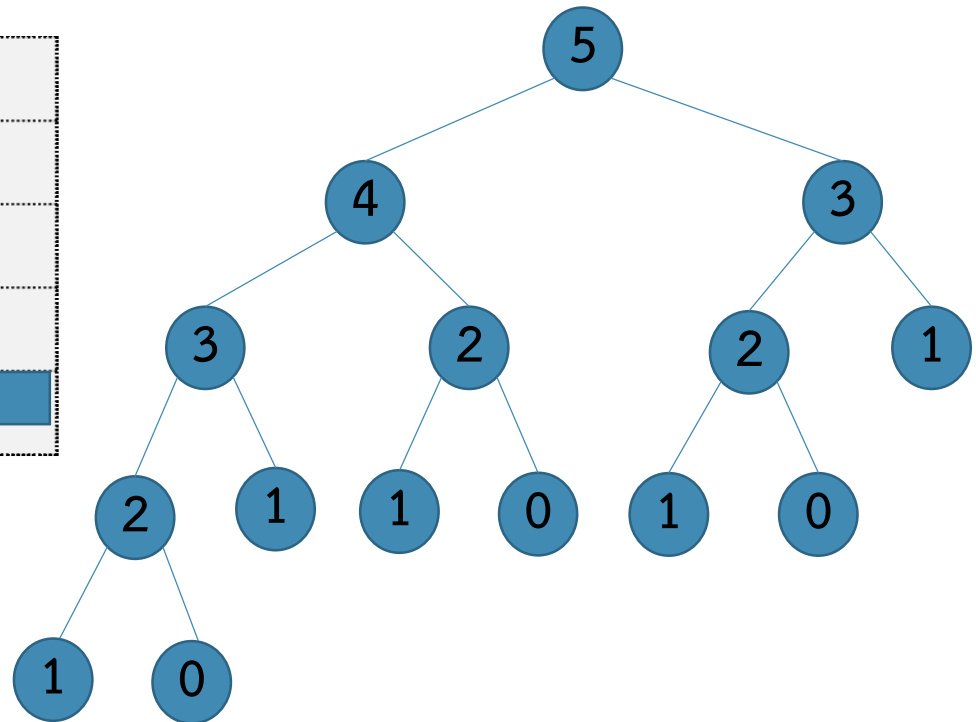
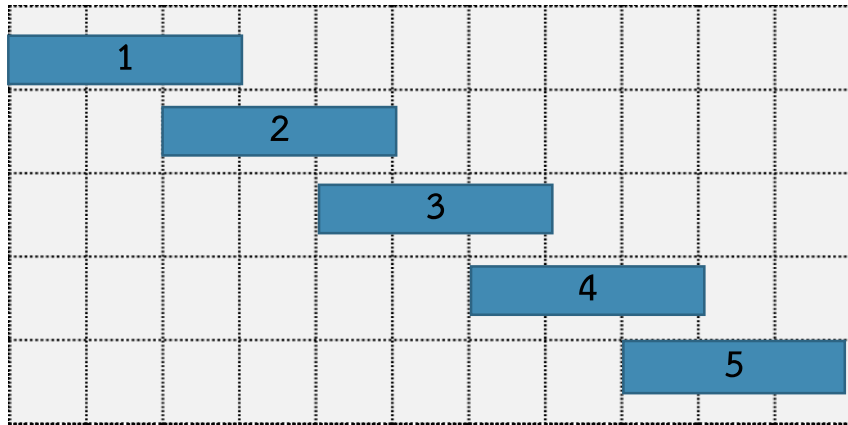
 return $\max(v_j + \text{Compute-opt}(p(j)), \text{Compute-opt}(j-1))$

}

Brute-force algorithm

จากการสังเกตเราพบว่าเมื่อใช้ recursive อย่างเดียวจะพบว่ามีปัญหาย่อยที่ซ้ำกันมาก
=> exponential algorithm

ตัวอย่างเช่น จำนวนของการเวียน recursive จะมีลักษณะโตคล้ายกับ 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 $S[n]$

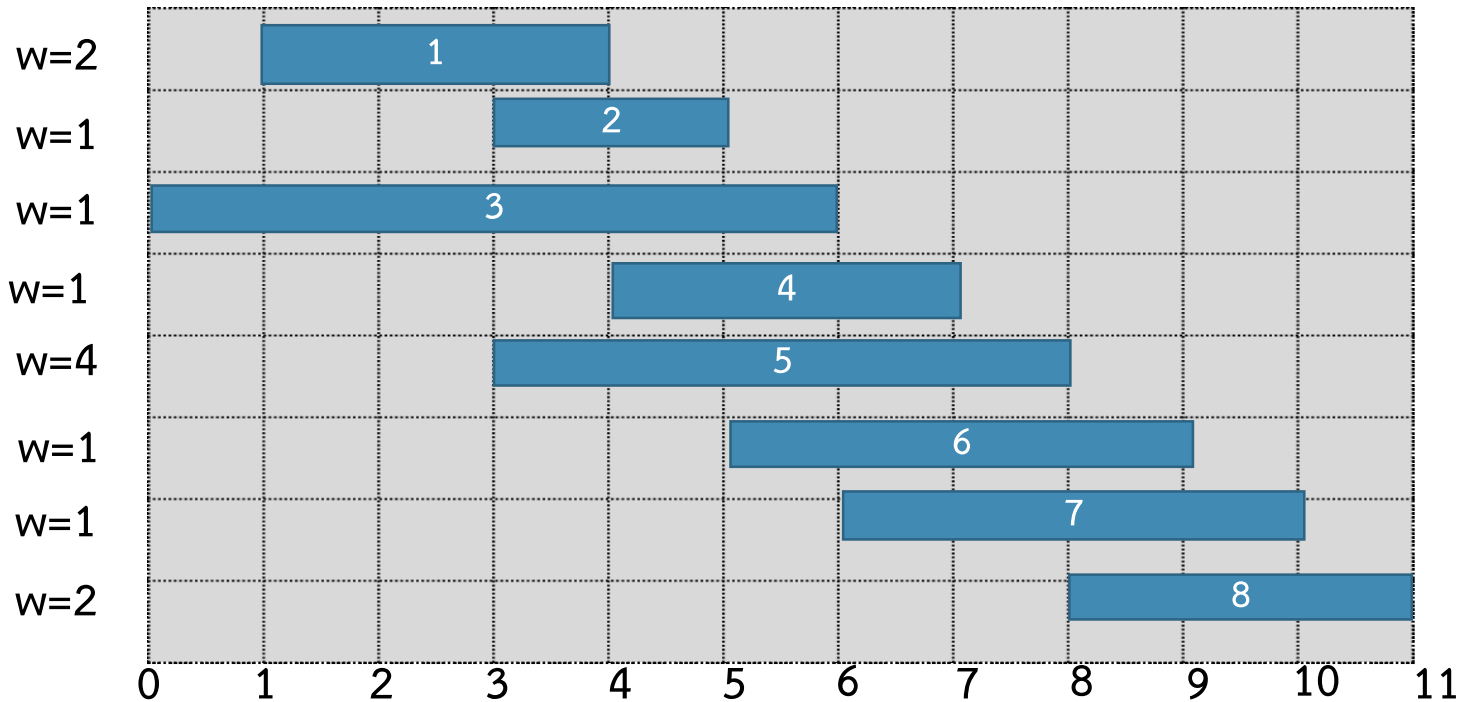
$S[j]$ = ค่าของคำตอบที่ดีที่สุดสำหรับงาน 1 ถึง j

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	