Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281
ผู้สอน: ตอน 1 อ. เบญจมาศ ปัญญางาม เรียน ห้อง 201
ตอน 2 อ. ดร. จักริน ชวชาติ เรียน ห้อง 209

วันสอบปลายภาค : วันเสาร์ที่ 7 ธ.ค. 2562
เวลา 8:00 -11:00 น. (ตามประกาศมหาวิทยาลัย)

บทที่ 9
Dynamic Programming Part III

0-1 Knapsack problem

Knapsack problem

กำหนดให้มีวัตถุหรือสิ่งของหลายๆ ชิ้น (Item[1..n]) ซึ่งสิ่งของแต่ละชิ้นมีน้ำหนัก (Weight[1..n]) และมีมูลค่า (Value[1..n]) อยู่

เราต้องการเอาสิ่งของต่างๆ ใส่ถุงเป้ โดยให้ได้ผลรวมราคาของสิ่งของในเป้มีค่ามากที่สุด ในขณะที่ผลรวมน้ำหนักของสิ่งของที่ถุงเป้รับได้จะต้องไม่เกินค่าบางค่า (W)

Item#	น้ำหนัก	มูลค่า
1	1	8
2	3	6
3	5	5

Knapsack problem

สำหรับปัญหานี้จะมี 2 version

1. 0-1 knapsack problem
2. Fractional knapsack problem (แบ่งของเป็นชิ้นย่อยๆ ได้)

แบบที่ 1 สิ่งของแบ่งย่อยไม่ได้: ดังนั้น

เราสามารถเลือกหยิบหรือไม่หยิบ

ปัญหานี้แก้ได้ด้วย Dynamic programming

แบบที่ 2 สิ่งของแบ่งย่อยได้: ดังนั้น

เราสามารถเลือกส่วนของสิ่งของมาได้

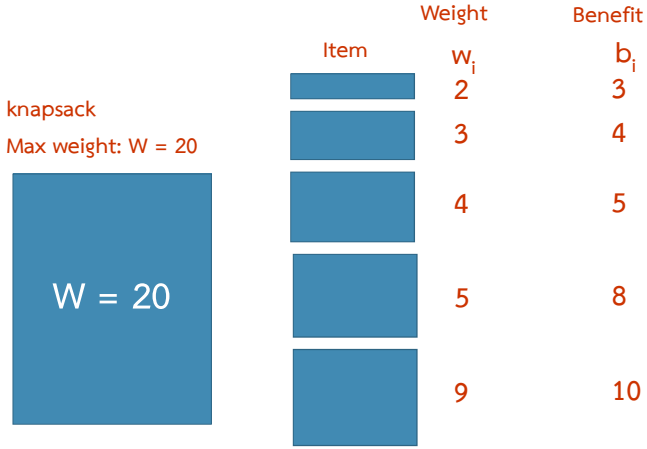
ปัญหานี้แก้ได้ด้วย Greedy algorithm (ทำได้ยังไง?)

0-1 Knapsack problem

กำหนดให้ ถุงผ้าที่มีความจุ W และเซต S ที่ประกอบด้วยสิ่งของ n ชิ้น โดยสิ่งของแต่ละชิ้น (ชิ้นที่ i) จะมีน้ำหนัก w_i และมีมูลค่า b_i (กำหนดให้ทุกๆ w_i, b_i และ W เป็นจำนวนเต็ม)

ปัญหา จะบรรจุสิ่งของลงถุงผ้าอย่างไรเพื่อให้สิ่งของที่ถูกบรรจุมีผลรวมมูลค่ามากสุด

0-1 Knapsack problem



Brute-force algorithm

- หากแก้ปัญหานี้แบบตรงๆ

เนื่องจากมีสิ่งของ n ชิ้น เพราะฉะนั้นจะมีรูปแบบที่เป็นไปได้ 2^n แบบ (สิ่งของหนึ่งชิ้นมี 2 ทางเลือกคือ เลือกเอาและเลือกไม่เอา)

เราจะลองทุกๆ รูปแบบนี้และหาแบบที่ได้มูลค่ารวมที่มากที่สุด โดยที่มีน้ำหนักรวมไม่เกิน W

ดังนั้นใช้เวลาในการทำงาน $O(2^n)$

คำถาม : จะทำได้ดีกว่านี้ไหม

คำตอบ : ได้ ใช้ dynamic programming

ดังนั้นเราต้องระบุ subproblem ให้ได้

Dynamic programming

1. **นิยาม subproblem**

หากกำหนดให้ชื่อสิ่งของเป็นหมายเลข 1, 2, ..., n แล้ว เราจะนิยาม subproblem จะเป็น

S_k = คำตอบที่ดีที่สุดในการเลือกสิ่งของหมายเลขที่ 1 จนถึงหมายเลขที่ k

เหมาะสมไหม

คำถาม: เราสามารถอธิบายคำตอบสุดท้าย S_n ได้ด้วย S_k หรือไม่

คำตอบ: ทำไม่ได้

Dynamic programming

นิยาม subproblem

ตัวอย่างกำหนดสิ่งของ 5 ชิ้น แต่ละชิ้นค่าน้ำหนักและมูลค่าดังตาราง และค่า Max weight ของถุงเป้เป็น 20

หากพิจารณาครบ 5 ชิ้น จะได้

- S_5 = คำตอบที่ดีที่สุดในการเลือกสิ่งของหมายเลขที่ 1 จนถึงหมายเลขที่ 5 นั่นคือจะได้ถุงเป้มีน้ำหนักรวม 20 และมูลค่ารวม 26

W_1=2	W_3=4	W_4=5	W_5=9
b_1=3	b_3=5	b_4=8	b_5=10

- คำถาม S_4 เป็นส่วนหนึ่งของคำตอบของ S_5 หรือไม่

Item	W_i	B_i
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

Dynamic programming

นิยาม subproblem

พิจารณา S_4: น้ำหนักรวม 14, มูลค่ารวม: 20

W_1=2	W_2=4	W_3=5	W_4=3
b_1=3	b_2=5	b_3=8	b_4=4

พิจารณา S_5: น้ำหนักรวม 20, มูลค่ารวม: 26

W_1=2	W_3=4	W_4=5	W_5=9
b_1=3	b_3=5	b_4=8	b_5=10

พบว่าคำตอบของ S_4 ไม่ได้เป็นส่วนหนึ่งของ S_5

Item	W_i	B_i
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

กรณีเลือกสิ่งของชิ้นที่ 5 จะไม่ได้ใช้คำตอบที่ดีที่สุด S_4 โดยตรงแต่ต้องพิจารณากรณีที่ถุงเป้ได้มูลค่ารวมสูงสุดแต่มีน้ำหนักรวมไม่เกิน 20

ดังนั้นจะต้องพิจารณาจากน้ำหนักรวมที่ถุงเป้รับได้ด้วย

Dynamic programming

เราพบว่าคำตอบของ S_4 ไม่ได้เป็นส่วนหนึ่งของคำตอบของ S_5

แสดงว่าการนิยาม S_k ยังไม่เพียงพอ แล้วทำอย่างไรดี

เราพบว่าจริงๆแล้วมี parameter 2 ตัวที่ต้องคำนึงถึง

ดังนั้นเราต้องเพิ่ม parameter อีกหนึ่งตัว คือ w ซึ่งแทนน้ำหนักจริงของแต่ละ subset ของสิ่งของ

นิยามใหม่ subproblem จะเป็นการคำนวณ B[k,w]

Dynamic programming

หา recurrence ของ subproblem

$$B[k,w] = \begin{cases} B[k-1,w] & if\ w_k > W \\ \max\{B[k-1,w], B[k-1,w-w_k] + b_k & \text{กรณีอื่นๆ} \end{cases}$$

หมายความว่า ซับเซตของสิ่งของที่ดีที่สุดของ S_k ที่มีน้ำหนักรวม w คือ

1. ซับเซตของสิ่งของที่ดีที่สุดของ S_{k-1} ที่มีน้ำหนักรวม w
2. ซับเซตของสิ่งของที่ดีที่สุดของ S_{k-1}ที่มีน้ำหนักรวม $w-w_k$ รวมกับสิ่งของใหม่ k

นั่นคือซับเซตที่ดีที่สุดของ S_k มีน้ำหนักรวม w มีการเลือกหรือไม่เลือกสิ่งของ k

กรณีแรก $w_k>w$ สิ่งของ k ไม่สามารถเป็นส่วนหนึ่งของคำตอบได้

กรณีที่สอง $w_k \leq w$ สิ่งของ k สามารถเป็นส่วนหนึ่งของคำตอบได้และเราจะเลือกกรณีที่ให้ค่ามากที่สุด

Dynamic programming

3. หา base case

1. หากมีสิ่งของแต่ถุงเป้มีขนาดเป็น 0 แสดงว่าเลือกได้ไหม,มูลค่ารวม=?

2. หากไม่มีของแต่ถุงเป้มีขนาดต่างๆ แสดงว่าเลือกได้ไหม,มูลค่ารวม=?

กรณีแรกคือ

```
for i=1 to n
    B[i,0]=0
```

กรณีที่สองคือ

```
for w=0 to W
    B[0,W]=0
```


Dynamic programming : Algorithm

```
for i=1 to n
    B[i,0]=0
for w=0 to W
    B[0,w]=0
for i=1 to n
    for w=0 to W
        if w_i <= w
            if b_i+B[i-1,w-w_i]>B[i-1,w]
                B[i,w] = b_i+B[i-1,w-w_i]
            else
                B[i,w] = B[i-1,w]
        else B[i,w] = B[i-1,w] //w_i > w
```


Dynamic programming : Example

n=4

W=5

สิ่งของ (น้ำหนัก, มูลค่า)

(2,3), (3,4), (4,5), (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0						
1						
2						
3						
4						

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

ใส่ค่า base case

```
for i=1 to n
    B[i,0]=0
for w=0 to W
    B[0,w]=0
```


Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=1$

$w-w_i=-1$

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3			
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=2$

$w-w_i=0$

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3		
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=3$

$w-w_i=1$

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=4

$w-w_i$=2

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=5

$w-w_i$=3

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0				
3	0					
4	0					

i=2

b_i=4

w_i=3

w=1

$w-w_i$=-2

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3			
3	0					
4	0					

i=2

b_i=4

w_i=3

w=2

$w-w_i$=-1

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4		
3	0					
4	0					

i=2

b_i=4

w_i=3

w=3

w-w_i=0

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	
3	0					
4	0					

i=2

b_i=4

w_i=3

w=4

w-w_i=1

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Example

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4	0					

i=2

b_i=4

w_i=3

w=5

w-w_i=2

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming : Practice

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4	0					

```
if w_i <= w
    if b_i+B[i-1,w-w_i]>B[i-1,w]
        B[i,w] = b_i+B[i-1,w-w_i]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

Dynamic programming

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

if w_i <= w
  if b_i+B[i-1,w-w_i]>B[i-1,w]
    B[i,w] = b_i+B[i-1,w-w_i]
  else
    B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

รู้ได้อย่างไรว่าเราใส่สิ่งของใดลงในเป้บ้าง

ตอนนี้ข้อมูลทุกอย่างอยู่ในตาราง

B[n,W] เป็นค่ามากที่สุดของสิ่งของที่ถูกใส่เข้าไปในถุงเป้

ให้ i=n และ k=W

if B[i,k] != B[i-1,k] then

  mark the i^{th} item as in the knapsack

  i=i-1, k=k-w_i

else

  i=i-1 //Assume the ith item is not in the knapsack

  //Could it be in the optimally packed knapsack?

ตัวอย่างการหาว่าเราใส่สิ่งของใดลงในเป้บ้าง

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=n, k=W
while i,k>0
  if B[i,k] != B[i-1,k] then
    mark the i^{th} item as in the knapsack
    i=i-1, k=k-w_i
  else
    i=i-1

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

ตัวอย่างการหาว่าเราใส่สิ่งของใดลงในเป้บ้าง

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=n, k=W
while i,k>0
  if B[i,k] != B[i-1,k] then
    mark the i^{th} item as in the knapsack
    i=i-1, k=k-w_i
  else
    i=i-1

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

33

ตัวอย่างการหาว่าเราใส่สิ่งของใดลงในเป้บ้าง

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

```
i=n, k=W
while i,k>0
    if B[i,k] != B[i-1,k] then
        mark the i^th item as in the knapsack
        i=i-1, k=k-w_i
    else
        i=i-1
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

34

ตัวอย่างการหาว่าเราใส่สิ่งของใดลงในเป้บ้าง

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

```
i=n, k=W
while i,k>0
    if B[i,k] != B[i-1,k] then
        mark the i^th item as in the knapsack
        i=i-1, k=k-w_i
    else
        i=i-1
```

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

35

ตัวอย่างการหาว่าเราใส่สิ่งของใดลงในเป้บ้าง

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

```
i=n, k=W
while i,k>0
    if B[i,k] != B[i-1,k] then
        mark the i^th item as in the knapsack
        i=i-1, k=k-w_i
    else
        i=i-1
```

The optimal knapsack should be {1,2}

Item(w,b)
1 (2,3)
2 (3,4)
3 (4,5)
4 (5,6)

36

Practice

- กำหนดให้มีถุงเป้ขนาด 8 หน่วย และมีสิ่งของ 4 ชิ้นที่มีมูลค่าและน้ำหนักดังนี้

Item#	มูลค่า	น้ำหนัก
1	15	1
2	10	5
3	9	3
4	5	4

- จงเลือกสิ่งของเพื่อให้ได้มูลค่ารวมสูงสุดที่มีน้ำหนักรวมไม่เกินน้ำหนักที่ถุงเป้รับได้และให้บอกน้ำหนักที่ได้ด้วย

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(จำนวนตัวอักขระ) มากที่สุด

ตัวอย่าง

S1 = AAACCGTGAGTTATTCGTTCTAGAA

S2 = CACCCCTAAGGTACCTTTGGTTC

ลองหา Longest common subsequence ของ S1 กับ S2

ตัวอย่าง Longest Common Subsequence

S1 = AAACCGTGAGTTATTCGTTCTAGAA

S2 = CACCCCTAAGGTACCTTTGGTTC

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] = |LCS(x[1..i],y[1..j])|

แล้ว c[m,n] = |LCS(x,y)|

Dynamic Programming

2. หา recurrence ของ subproblem

หากกำลังพิจารณา string x ถึงตำแหน่งอักขระที่ i

และพิจารณา string y ถึงตำแหน่งอักขระที่ j

กำหนดให้ Z = $z_1z_2...z_p$ 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] != 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 และอักขระใน string y ตัวที่ j เหมือนกัน

เช่น กำหนด x=BDCABA และ y=ABCBDAB

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

- จะพบว่า $z_1z_2...z_{p-1}$ เป็นคำตอบที่ดีที่สุดของ x[1..i-1] ("BDCA") และ y[1..j-1] ("ABC") นั่นคือ มี "BC" เป็น LCS
- ดังนั้น คำตอบที่ดีที่สุดของ x[1..i] ("BDCAB") และ y[1..j] ("ABCB")
- 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] ≠ y[j] หมายถึงกรณีอักขระใน string x ตัวที่ i และอักขระใน string y ตัวที่ j ไม่เหมือนกัน

เช่น กำหนด x=BDCABA และ y=ABCBDAB

หากกำลังพิจารณาตำแหน่งที่ i=6 และ j=4 นั่นคือ x[6] ≠ y[4]

- พบว่าความยาวของคำตอบที่ดีที่สุดไม่เพิ่มขึ้น
- ดังนั้น $z_1z_2...z_p$ 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])

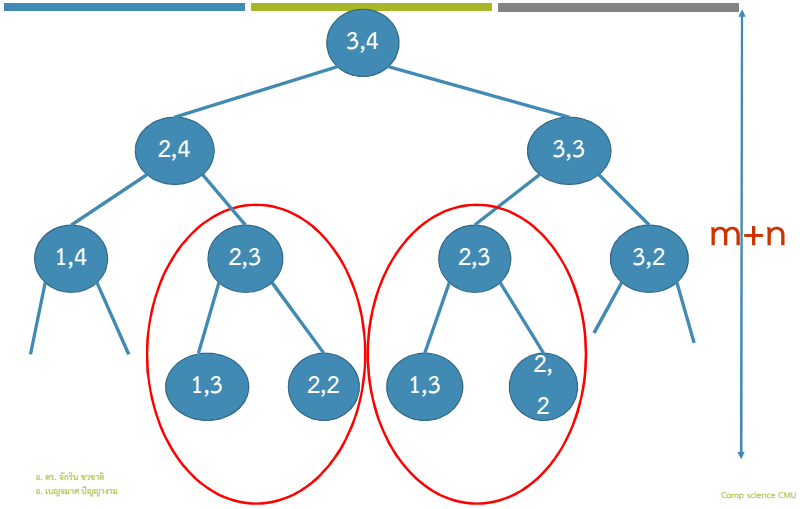
Dynamic Programming : หา recurrence

- เขียนสมการ recurrence ได้ดังนี้

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

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 ไม่มีสักตัว (ยาว=0 ตัว) LCS ควรเป็นเท่าไร

```
for i=1 to m
    c[i,0] = 0
```

ถ้า string x ไม่มีสักตัว (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=ABCBDAB

		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[1]!=y[1]

- X=BDCABA
- Y=ABCBDAB

		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]!=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=ABCBDAB

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=ABCBDAB

x[1]!=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]!=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=BDCABA
- Y=ABCBDAB

x[1]=y[7]

		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							
C	0							
A	0							
B	0							
A	0							

Dynamic programming : Example

- X=BDCABA
- Y=ABCBDAB

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 จนครบ

- 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 จนครบ

สร้างตารางย้อนกลับ
เหมือนกัน ให้ใส่ ↖
หากเลือกจากด้านบนให้ใส่ ↑
หากเลือกจากด้านซ้ายให้ใส่ ←
จุดที่เป็น ↖ คือคำตอบ

- 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

CS 204451
บทที่ 9

Fill จนครบ

สร้างตารางย้อนกลับ
เหมือนกัน ให้ใส่
หากเลือกจากด้านบนให้ใส่
หากเลือกจากด้านซ้ายให้ใส่
จุดที่เป็น คือคำตอบ

- 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

อ. ดร. จักริน ชวชาติ
อ. เบญจมาศ ปัญญางาม
Comp science CMU