Dynamic Programming

Algorithmic Paradigms

- Greedy
  - จะค่อยๆ สร้างคำตอบขึ้นมา โดยจะเลือกทำสิ่งที่ดีที่สุดในแต่ละรอบของการตัดสินใจเพื่อเลือกคำตอบ
- Divide and Conquer
  - จะแบ่งปัญหาออกเป็นปัญหาย่อย โดยแบ่งไปเรื่อยๆ จนเป็นปัญหาที่ง่ายแล้วแก้
  - จากนั้นค่อยๆ รวมคำตอบของปัญหาย่อยนั้นกลับขึ้นมาเป็นคำตอบของปัญหาตั้งต้น
- Dynamic Programming
  - จะแบ่งปัญหาออกเป็นลำดับของปัญหาย่อย**ที่ซ้ำกัน**
  - คำนวณแล้วเก็บคำตอบไว้เพื่อที่จะได้ไม่ต้องคำนวณใหม่ทั้งหมด
  - จากนั้นนำคำตอบของปัญหาย่อยมาสร้างเป็นคำตอบของปัญหาตั้งต้น

Coin Change

ตัวอย่าง

สมมติว่า มีเหรียญ 1, 4, 5, 10 บาท

($d_0 = 1$, $d_1 = 4$, $d_2 = 5$, $d_3 = 10$)

ต้องการทอนเงิน 7 บาท ใช้เหรียญอะไรบ้าง

5, 1, 1

ต้องการทอนเงิน 8 บาท ใช้เหรียญอะไรบ้าง

4, 4

เราจะจัดการปัญหานี้อย่างไร

Dynamic programming

Steps ในการแก้ปัญหาด้วยเทคนิค Dynamic programming

1. นิยาม subproblem
2. หา recurrence ของ subproblem
   - เป็นการหาวิธีการรวมคำตอบของ subproblem ให้เป็นคำตอบที่ใหญ่ขึ้น
3. แสดงและแก้ base case

Coin change : Dynamic programming

1. **นิยาม** subproblem

ให้ C[p] แทนจำนวนเหรียญที่**น้อย**ที่สุดที่ต้องการในการแลกเงิน p บาท

Coin change : Dynamic programming

1. **นิยาม** subproblem

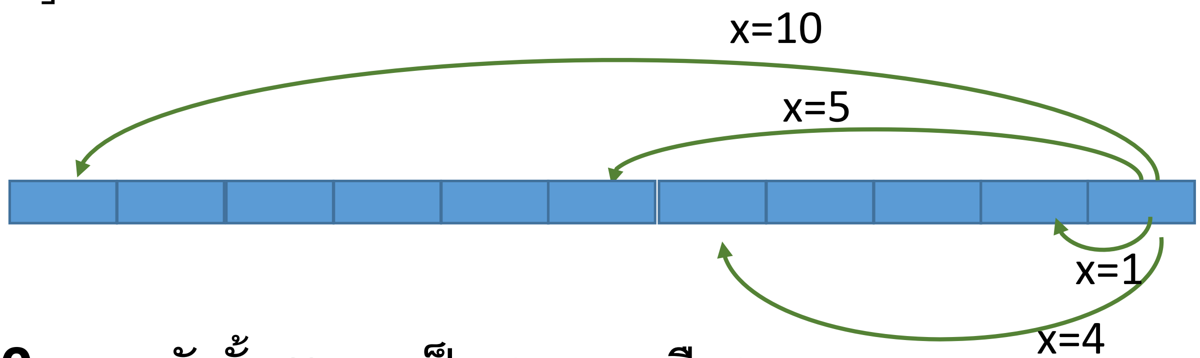
ให้ C[p] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

หา recurrence **ของ** subproblem

ให้ x แทนค่าของเหรียญที่ถูกใช้ในคำตอบที่ดีที่สุด ขณะแลกเงิน p บาท

ดังนั้น C[p] = 1 + C[p-x]

ปัญหา แต่เราไม่รู้ค่า x



สมมติว่า มีเหรียญ 1, 4, 5, 10 บาท ดังนั้น x อาจเป็น 1 4 5 **หรือ**10 **บาท**

หมายเหตุ C[p-x] จำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p-x บาท

➔ **เป็นคำตอบที่ดีที่สุดในการแลกเงิน p-x บาท**

Coin change : Dynamic programming

1. **นิยาม** subproblem

   ให้ C[p] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

   หา recurrence **ของ** subproblem

   ให้ x แทนค่าของเหรียญอันแรกที่ถูกใช้ในคำตอบที่ดีที่สุด

   ดังนั้น C[p] = 1 + C[p-x]

   ปัญหา แต่เราไม่รู้ค่า x

   คำตอบ เราจะลองทุกๆ ค่า x แล้วใช้ค่าจำนวนต่ำสุด (คำตอบที่ดีที่สุด)

2. **หา** recurrence **ของ** subproblem

$$C[p] = \min_{i:d_i \le p} \{C[p-d_i] + 1\}$$

โดย ให้ d_i แทนเหรียญที่ i

หา base case

สมมติว่ามีแค่เหรียญ 1, 4, 5 และ 10

($d_0 = 1, d_1 = 4, d_2 = 5, d_3 = 10$)

Coin change : Dynamic programming

1. **นิยาม** subproblem

   ให้ C[p] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

   หา recurrence **ของ** subproblem

   ให้ x แทนค่าของเหรียญอันแรกที่ถูกใช้ในคำตอบที่ดีที่สุด

   ดังนั้น C[p] = 1 + C[p-x]

   ปัญหา แต่เราไม่รู้ค่า x

   คำตอบ เราจะลองทุกๆ ค่า x แล้วใช้ค่าจำนวนต่ำสุด (คำตอบที่ดีที่สุด)

2. **หา** recurrence **ของ** subproblem

$$C[p] = \min_{i:d_i \leq p} \{C[p-d_i] + 1\}$$

   โดย ให้ d_i แทนเหรียญที่ i

3. **หา** base case C[0] =0

สมมติว่ามีแค่เหรียญ 1, 4, 5 และ 10

$$C[p] = \begin{cases} min_{i:d_i \leq p}\{C[p-d_i]+1\} & if\ p > 0 \\ 0 & if\ p = 0 \end{cases}$$

โครงสร้างของ pseudocode ในการแก้ปัญหา Dynamic programming

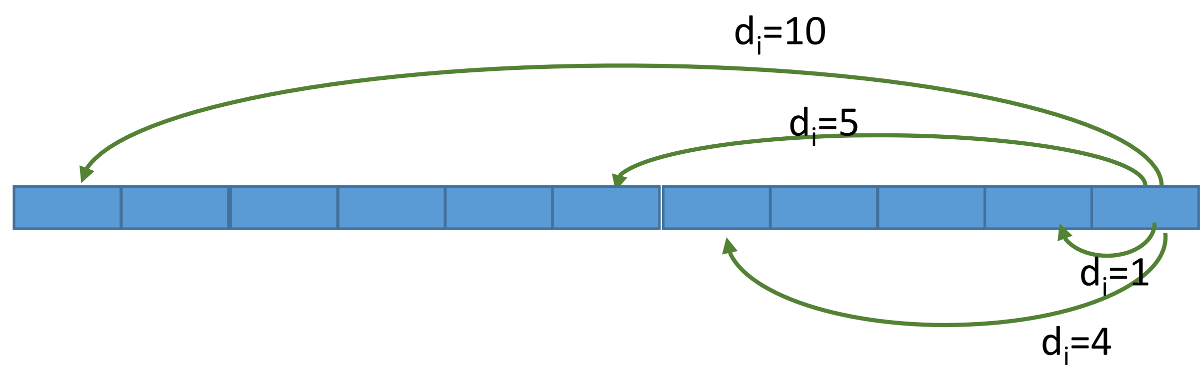
```
Solution DynamicAlgo(s){
    if (s==basecase) return basecase_solution
    if (memo.contain(s)) return memo.get(s)
    Solution ans = recurrence_relation(s)
    memo.put(s, ans)
     return ans
}
```

Dynamic programming algorithm


```
void CHANGE(int n){
	int C[n], min;
	int d[4] = {1, 4, 5, 10}, k = 4;
	C[0] = 0;				//base case
	for(int p = 1; p <= n; p++){
		min = n;
		for (int i = 0; i < k; i++){
			if (p >= d[i]){
				if(C[p - d[i]] + 1 < min){
					min = C[p - d[i]] + 1;
				}
			}
		}
		C[p] = min;
	}
	return C;
}
```

สมมติว่ามีแค่เหรียญ 1, 4, 5 และ 10

($d_0 = 1, d_1 = 4, d_2 = 5, d_3 = 10$)



ตัวอย่างการค่าคำตอบในแต่ละปัญหาย่อย

C

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	2	3	1	1	2	3	2	2	1	2	3	3	2

d

0	1
1	4
2	5
3	10

เริ่มจาก 1 บาทไปจนถึง n

1 บาทลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท

(ถามจากค่าที่ดีสุดในการแลกเงิน 0 บาท + 1)

หมายถึงเลือกเหรียญ 1 บาทดังนั้น C[1]=1+C[1-1]

ซึ่งจะได้คำตอบที่ดีที่สุดในการแลกเงิน 1 บาท เป็นจำนวน 1 เหรียญ (1+0)

2 บาทลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท

(ถามจากค่าที่ดีสุดในการแลกเงิน 1 บาท + 1)

3 บาทลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท

(ถามจากค่าที่ดีสุดในการแลกเงิน 2 บาท + 1)

4 บาทลองแทนว่าใช้เหรียญอะไรทอนได้

พบว่าใช้เหรียญบาท 4 เหรียญ (ถามจากค่าที่ดีสุดในการแลกเงิน 3 บาท + 1)

หรือพบว่าใช้เหรียญ 4 บาท 1 เหรียญ (ถามจากค่าที่ดีสุดในการแลกเงิน 0 บาท + 1)

แบบฝึกหัด

- กำหนดให้มีเหรียญ 1, 5, 7, 10 บาท จงวาดตาราง C ที่แสดงจำนวนในการทอนเหรียญ 20 บาท

Find the number of different ways to write n

Find the number of different ways to write n

กำหนด ค่า n และเซตของตัวเลข= {1, 3, 4}

Goal: ต้องการจำนวนวิธีที่ต่างกันในการเขียนค่า n โดยการใช้การรวมกันของตัวเลข 1, 3, 4

ตัวอย่าง

n = 5 มี 6 วิธี

$$
\begin{aligned}
5 &= 1 + 1 + 1 + 1 + 1 \\
&= 1 + 1 + 3 \\
&= 1 + 3 + 1 \\
&= 3 + 1 + 1 \\
&= 1 + 4 \\
&= 4 + 1
\end{aligned}
$$

Solution by Dynamic programming

1. **นิยาม** sub problem

ให้ D_n แทนจำนวนวิธีในการเขียน n เมื่อเป็นผลรวมของ 1, 3, 4

Solution by Dynamic programming

1. **นิยาม** sub problem

ให้ D_n แทนจำนวนวิธีในการเขียน n เมื่อเป็นผลรวมของ 1, 3, 4

2. **หา** recurrence

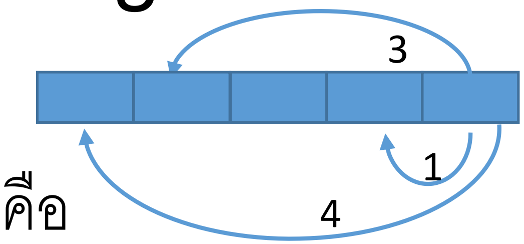
พิจารณารูปแบบของคำตอบที่เป็นไปได้แบบหนึ่ง $n = x_1+x_2+\ldots+ x_m$

ถ้า x_m เป็น 1 ส่วนที่เหลือจะต้องรวมกันให้ได้ n-1

นั่นคือจำนวนของรูปแบบที่ตัวสุดท้ายที่เลือกคือ $x_m = 1$ จะเท่ากับ D_{n-1}

สำหรับกรณีอื่น ($x_m=3$, $x_m=4$) ก็คิดแบบเดียวกัน

Solution by Dynamic programming



เนื่องจากค่า n เป็นผลรวมของ 1, 3, 4

จึงพบว่ารูปแบบในการเขียน n เกิดได้จาก 3 รูปแบบ คือ

1) หากตัวสุดท้าย (x_m) เขียนด้วย 1 จำนวนวิธีจะเท่ากับ D_{n-1}

2) หากตัวสุดท้ายเขียนด้วย 3 จำนวนวิธีจะเท่ากับ D_{n-3}

3) หากตัวสุดท้ายเขียนด้วย 4 จำนวนวิธีจะเท่ากับ D_{n-4}

ดังนั้นคำตอบของจำนวนวิธีในการเขียน n จะได้จากนำทุกวิธีมารวมกัน

จากตัวอย่าง **n = 5** มี **6** วิธี

5 = 1 + 1 + 1 + 1 + 1
= 1 + 3 + 1
= 3 + 1 + 1
= 4 + 1
= 1 + 1 + 3
= 1 + 4

คำตอบ D_5 จำนวนวิธีในการเขียน 5 ได้จาก

จำนวนวิธีของ D_4(กรณี $x_m = 1$)

\+ จำนวนวิธีของ D_2(กรณี $x_m = 3$)

\+ จำนวนวิธีของ D_1(กรณี $x_m = 4$)

ดังนั้น $D_5 = 1 + 1 + 4$

Solution by Dynamic programming

2. **หา** recurrence

$D_n = D_{n-1} + D_{n-3} + D_{n-4}$

3. **จัดการกรณี** base case

$D_0 = 1$

$D_n = 0$ ถ้า n มีค่าน้อยกว่า 0

กรณีอื่นๆ $D_0 = D_1 = D_2 = 1, D_3 = 2$

ตรวจสอบว่า base case ครบหรือยัง

ให้เทียบกับสมการเริ่มต้นว่าเริ่มทำงานได้หรือยัง

$\mathbf{D_4 = D_3 + D_1 + D_0}$

$\mathbf{D_3 = D_2 + D_0 + D_{-1}}$

$\mathbf{D_2 = D_1 + D_{-1} + D_{-2}}$

$\mathbf{D_1 = D_0 + D_{-2} + D_{-3}}$

หาก **n=3** จะมี 2 รูปแบบคือ 1+1+1 และ 3

หาก **n=2** จะมี 1 รูปแบบ คือ 1+1

หาก **n=1** จะมี 1 รูปแบบ คือ 1

ตั้งแต่ n=4 (D_3) ไม่มีค่าติดลบ ดังนั้น base case คือ D_3 D_2 D_1 และ D_0

Solution by Dynamic programming

```
int DifferentWaysToWriteN (int n) {
        D[0] = D[1] =D[2] = 1
        D[3] = 2
        for(i=4; i<=n; i++) {
                D[i] = D[i-1] + D[i-3] + D[i-4]
}
return(D[n])
```

Running Time เป็นเท่าไร

Maximum Value Contiguous Subsequence

Maximum Value Contiguous Subsequence

กำหนดให้ ลำดับของเลขจำนวนจริง (มีทั้งเลขบวกและเลขลบ) n ตัว A(1), A(2),…A(n)

สิ่งที่ต้องการ (Goal) หาลำดับที่ติดกัน A(i), …, A(j) ที่ผลรวมของสมาชิกในลำดับนั้นมีค่ามากที่สุด

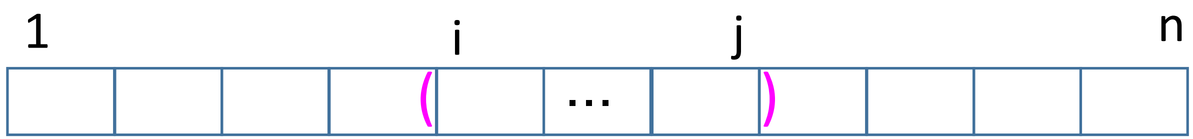
ตัวอย่างเช่น

- กำหนดลำดับของเลขเป็น 1, 3, 5, -4, 2

  จะได้ค่าผลรวมมากสุดเป็น 9 (เมื่อเลือกลำดับที่ติดกันเป็น 1, 3, 5)
- กำหนดลำดับของเลขเป็น 1, -5, 2, -1, 3 จะได้ค่าผลรวมมากสุดเท่าไร?

Solution by Dynamic programming

Input : array A[1.. n] **ของเลขจำนวนจริง**



Goal: $Max \sum_{x=i}^{j} A[x]$

1. **นิยาม** subproblem

ให้ B[j] แทนผลรวมที่ติดกันที่มากสุดเมื่อพิจารณาถึงช่องที่ j

Solution by Dynamic programming

2. **หา** recurrence **ของ** subproblem

เนื่องจาก**สิ่งที่ต้องการ**คือ หาลำดับที่ติดกัน A(i), …, A(j) ที่ให้ผลรวมมากที่สุด

ดังนั้นพิจารณาตัวที่ j

คำถาม: เมื่อพิจารณาตัวเลขตัวที่ j สิ่งที่เกิดขึ้นได้มีอะไรบ้าง

คำตอบ: จะตัดสินใจเลือกรวมตัวที่ j หรือไม่รวมตัวที่ j

คำถาม: การเลือกรวมตัวที่ j เพราะเหตุผลอะไร

คำตอบ: เมื่อนำผลรวมนับรวมตัวที่ j แล้วทำให้ผลรวมมีค่ามากขึ้น
โดย**มากกว่า** การเริ่มต้นนับใหม่ที่ตำแหน่งที่ j

คำถาม: การเลือกที่จะไม่รวมตัวที่ j เพราะเหตุผลอะไร

คำตอบ: การเริ่มต้นนับใหม่จะทำให้ได้ผลรวมมากกว่า

Solution by Dynamic programming

2. หา recurrence ของ subproblem

ตัวอย่างกำหนดลำดับของเลขเป็น **1, -5, 2, -1, 3** จะได้ค่าผลรวมมากสุดเท่าไร**?**

ค่า **j**	ค่า**A[j]**	ผลรวมลำดับติดกัน
0	1	มีตัวเดียว1
1	-5	จะเลือกระหว่าง 1+ -5 +? และ -5 +?
		คำตอบคือเลือก 1+ -5
		เพราะการนับรวมตัวที่ **j** แล้วทำให้ผลรวมมีค่ามากขึ้น
2	2	จะเลือกระหว่าง 1+-5+2 +? และ 2 +?
		คำตอบคือเลือก 2
		เพราะการนับรวมตัวที่ **j** แล้วไม่ได้ทำให้ผลรวมมีค่ามากขึ้น
3	-1	จะเลือกระหว่าง 2+ -1 +? และ -1 +?
		คำตอบคือเลือก 2+ -1

Solution by Dynamic programming

2. **หา** recurrence **ของ** subproblem

$$B[j] = \max\{B[j-1] + A[j] ,\ \ A[j]\}$$

โดยที่

B[j] = B[j-1] + A[j] คือ ผลรวมที่ติดกันที่มากสุดกรณีที่ขยายช่วงของคำตอบมารวมช่องที่ j

B[j] = A[j] คือ ผลรวมที่ติดกันที่มากสุดกรณีที่การเริ่มต้นนับใหม่ที่ตำแหน่งที่ j (A[j]) เป็นต้นไป

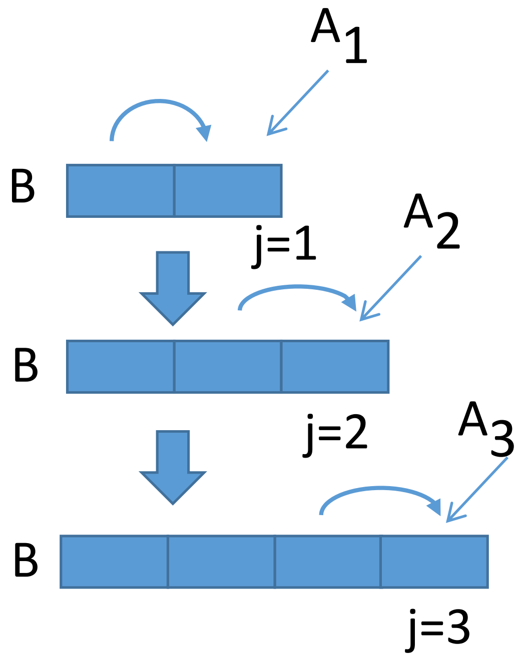
3. **หา** base case

ถ้ามีตัวเดียวก็ต้องตอบเลย ไม่ว่าจะติดลบหรือไม่ก็ตาม

B[0] = A[0]

Dynamic programming algorithm

```
int maxContiguousSum(int A[], int len) {
        int j;
        int B[len];
        B[0] = A[0];
        for (j = 1; j < len; j++) {
                B[j] = max(B[j-1]+A[j], A[j] );
        }
        int max_so_far = B[0];
        for (j = 1; j < len; j++) {
                if( max_so_far < B[j])
                        max_so_far = B[j];
        }
        return max_so_far;
}
```



Dynamic programming algorithm(Update version)

```
int maxContiguousSum(int A[], int n)
{       int j;
        int max_so_far = A[0], i;
        int curr_max = A[0];
        for (j = 1; j < n; j++)    {
                curr_max = max(curr_max + A[j] , A[j]);
                max_so_far = max(curr_max, max_so_far );
        }
        return max_so_far;
}
```

แบบฝึกหัด

- -2, 11, -4, 13, -5, 2

- -15, 29, -36, 3, -22, 11, 19, -5

Longest Increasing Subsequence

Longest Increasing Subsequence

กำหนดให้ ลำดับของตัวเลข A1, A2, … An

Goal ต้องการหา subsequence (ส่วนของลำดับไม่จำเป็นต้องติดกัน) ที่<u>**เพิ่มขึ้น**</u>ที่ยาวที่สุด

ตัวอย่างเช่น

ลำดับของตัวเลขคือ 1, 5, 3, 4, 8, 2, 6, 7

จะได้ subsequence ที่เพิ่มขึ้นที่ยาวที่สุดคือ 5 ตัว

ได้แก่ 1, 3, 4, 6, 7

Solution by Dynamic programming

1. **นิยาม** sub problem

ให้ L(j) ผลการนับ subsequence ที่เพิ่มขึ้น ที่ยาวที่สุด ณ ตัวที่ j

Solution by Dynamic programming

1. **นิยาม** sub problem

ให้ L(j) แทน ผลการนับ subsequence ที่เพิ่มขึ้นที่ยาวที่สุด ณ ตัวที่ j

2. **หา** recurrence **ของ** subproblem

พิจารณาตัวที่ j

คำถาม ตัวที่ j ควรจะนับต่อจากตัวไหน

คำตอบ ตัวที่มีค่าน้อยกว่ามันและมีผลการนับที่มากที่สุด

Solution by Dynamic programming

L(j) แทน ผลการนับ subsequence ที่เพิ่มขึ้นที่ยาวที่สุด ณ ตัวที่ j

ตัวอย่างลำดับของตัวเลขคือ 1,5, 3, 4, 8, 2, 6, 7

พิจารณาถึงตัวที่ j

j A(j) L(j)//พิจารณาทุกตัวที่ A[i]< A(j),i=0…j-1 แล้วเก็บค่ามากสุด

0 1 1 //ตัวแรกนับได้ 1 ตัว **1**

1 5 2 //เลือก **max** โดยคิดจากกรณี **L(0) +1➔ 2**

2 3 2 //เลือก **max** โดยคิดจากกรณี **L(0) +1➔ 2**

3 4 3 //เลือก **max** โดยคิดจากกรณี **L(0) +1➔2, L(2) +1➔3**

เพราะว่า **A[0] , A[2]< A(3)**

4 8 4 //เลือก **max** โดยคิดจากกรณี **L(0) +1 ➔2, L(1) +1➔ 3, L(2) + 1 ➔ 3, L(3) + 1 ➔ 4** เพราะทุกตัว**< A(4)**

5 2 2 //L(5) เลือก **max** โดยคิดจากกรณี **L(0) + 1 ➔ 2**

Solution by Dynamic programming

2.**หา** recurrence **ของ** subproblem

$$L(j) = \max_{i<j:A[i]<A[j]}\{L(i)\} + 1$$

เมื่อ $\max_{i<j:A[i]<A[j]}\{L(i)\}$ **คือ** L(i) **ที่มากที่สุดที่พิจารณาจากลำดับที่** i **ที่น้อยกว่า** j **ที่** A[i] < A[j]

3. **หา** base case

$$L(1) = 1$$

อย่าลืม เราจะหาค่ามากสุด แต่ว่าตอนนี้เราดูที่ว่าถ้านับเอาตัวที่ j แล้วจะทำให้มากที่สุดอย่างไร ไม่อย่างนั้นต้องวนหาค่ามากสุดอีกครั้งด้วย

Dynamic programming algorithm

```
int LIS(A[], n){
        for (i = 1 to n) L[i]=1
        max = L[1]
        for (i = 2 to n) {
                for (j = 1 to i - 1) {
                        if (A[i] > A[j] && L[i] < L[j] + 1)
                                L[i] = L[j] + 1
                                if(max<L[i])
                                        max=L[i]
                }
        }
        return max
}
```