Dynamic Programming

Algorithmic Paradigms

- Greedy จะค่อยๆ สร้างคำตอบขึ้นมา โดยจะเลือกทำสิ่งที่ดีที่สุดในแต่ละรอบ

- Divide and Conquer จะแบ่งปัญหาออกเป็นปัญหาย่อย แบ่งไปเรื่อยๆ จนปัญหาแก้ง่ายแล้วแก้ จากนั้นค่อยๆ รวมคำตอบของปัญหาย่อยนั้นกลับขึ้นมาเป็นคำตอบของปัญหาตั้งต้น

- Dynamic Programming จะแบ่งปัญหาออกเป็นลำดับของปัญหาย่อย**ที่ซ้ำกัน** คำนวณแล้วเก็บคำตอบไว้เพื่อที่จะได้ไม่ต้องคำนวณใหม่ทั้งหมด จากนั้นนำคำตอบของปัญหาย่อยมาสร้างเป็นคำตอบของปัญหาตั้งต้น

Fibonacci Numbers

ก่อนที่จะใช้เทคนิคของ dynamic programming นั้น จะแสดงเทคนิคที่เกี่ยวข้องที่เรียกว่า memoization กับปัญหาการหา Fibonacci number ตัวที่ n

นิยาม ของ Fibonacci number ตัวที่ n แทนด้วย F_n

$F_0 = 0$

$F_1 = 1$

$F_n = F_{n-1} + F_{n-2}$

Fibonacci Numbers

เริ่มต้นเราจะเขียน algorithm แบบธรรมดาในการหา Fibonacci number ซึ่งจะเขียนตามนิยาม

```
int fibo(int n) {
    if(n==0)  return 0;
    if(n==1)  return 1;
    return fibo(n-1) + fibo(n-2);
}
```

Running Time

$T(n) = T(n-1) + T(n-2) + c$

$\geq 2T(n-2) + c$ // เปลี่ยนเป็นวิเคราะห์อสมการนี้แทน

$\geq 2(2T(n-4) + c) + c$

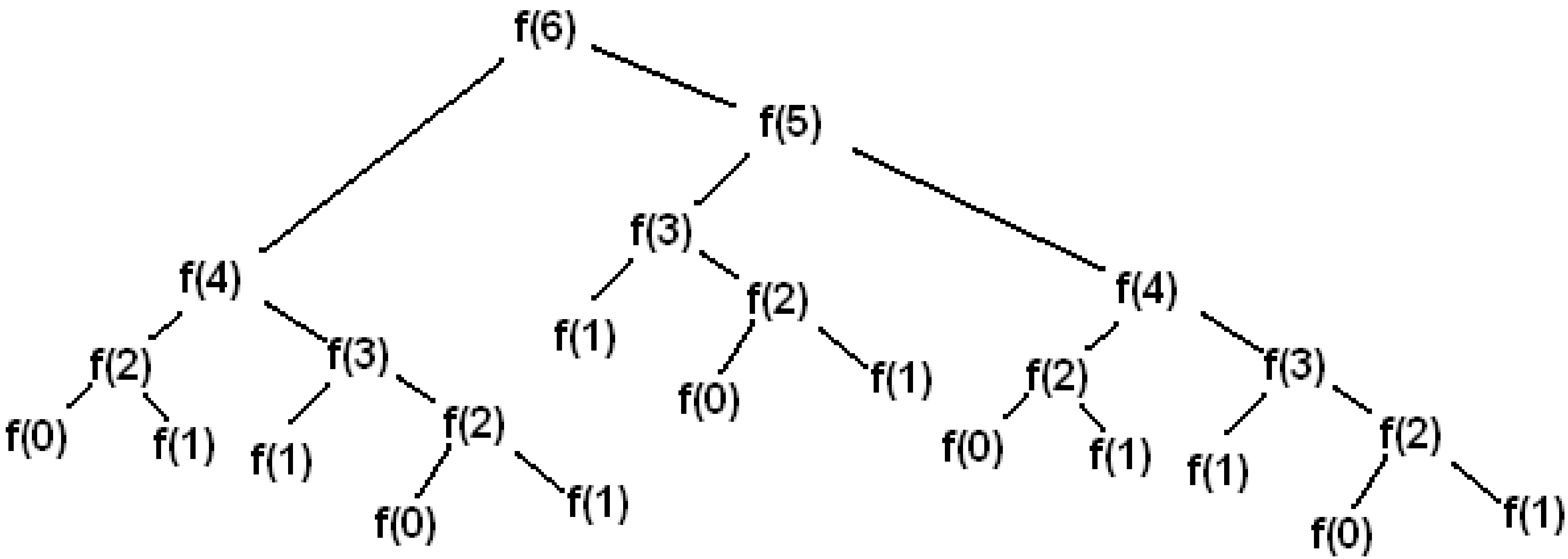
$\geq 4T(n-4) + 2c + c$

$\geq 4(2(T(n-6)+c) + 2c + c$

$\geq 8T(n-6) +4c + 2c + c$

$\geq 2^k(T-2k)+c(2^{k-1}+2^{k-2}+\ldots+2+1) = \Omega(c2^{n/2})$

ตัวอย่างการทำงาน



<u>ข้อสังเกต</u>

มีการคำนวณซ้ำกันหลายจุดมาก

โหนดใบของต้นไม้นี้จะเป็น 0 หรือ 1

ผลรวมของค่าเหล่านี้จะถูกรวมขึ้นไปกลายเป็นผลลัพธ์สุดท้าย

Memoization Methodlogy

สังเกตได้ว่าหลายๆ ปัญหาย่อยมีการคำนวณที่เหมือนกัน

เพื่อเป็นการทำให้ไม่เสียเวลาในการคำนวณ เราจะคำนวณปัญหาย่อยเหล่านั้นแล้วเก็บผลลัพธ์ไว้ในตาราง เช่น array

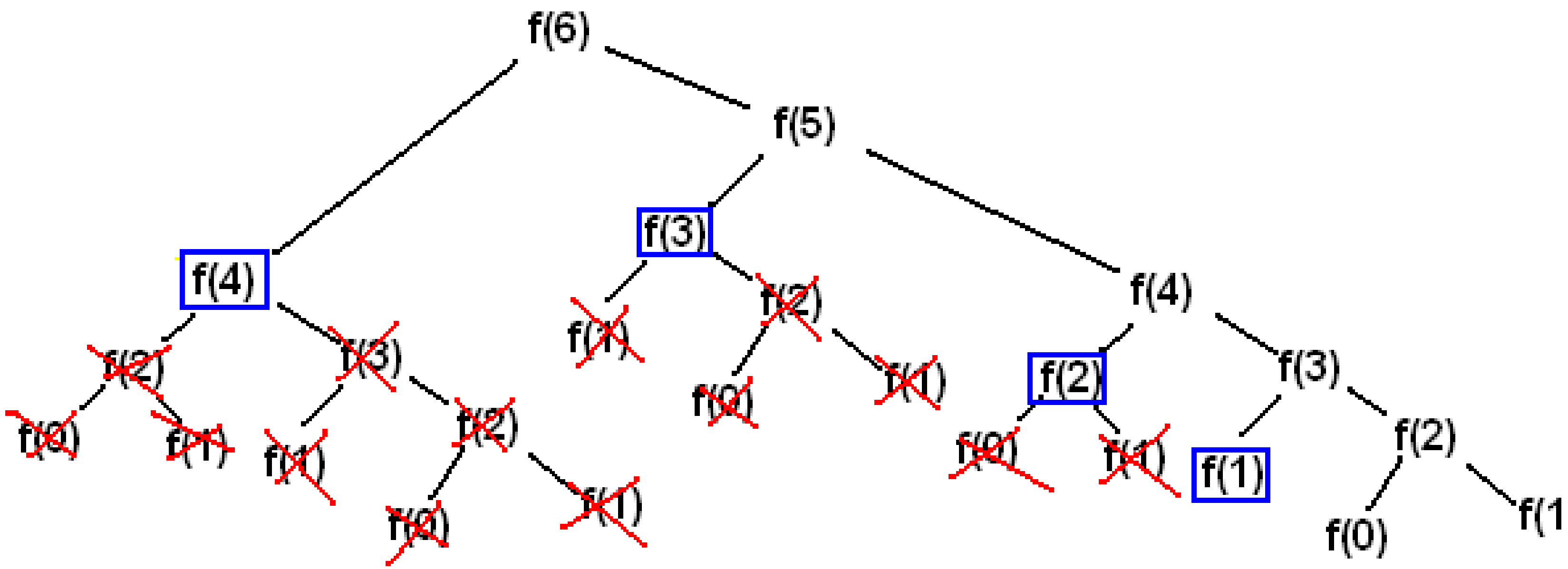
เมื่อต้องการแก้ปัญหาย่อยอีกครั้งเราจะนำเอาผลลัพธ์ที่คำนวณไว้แล้วตอบได้เลย

Memoization Methodlogy

1. มองปัญหาแบบย้อนกลับ
2. ค้นหาผลลัพธ์ของปัญหาย่อยในตาราง
3. ถ้าไม่พบในตาราง คำนวณแบบ recursive ไปแล้วเก็บผลลัพธ์ที่ได้ไว้ในตาราง ก่อน return ค่า

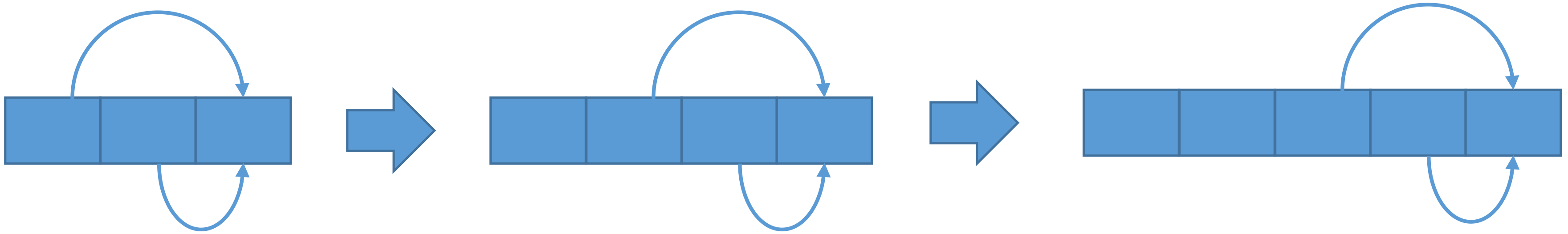
ข้อสังเกต

เราสามารถเปลี่ยนอัลกอริทึมที่มีการ memoization แบบ recursive นี้มาเป็นอัลกอริทึมแบบทำซ้ำจากล่างขึ้นบน (bottom-up) ได้ โดยการเติมคำตอบของปัญหาย่อยลงในตาราง



หากค่าในกล่องสี่เหลี่ยมจะถูกคำนวณไว้แล้ว และถูกนำมาใช้ เห็นได้ชัดว่าอัลกอริทึมจะเร็วกว่าการคิดแบบตรงๆ (คำนวณทั้งหมด)

```
int fibo(int n) {
        T[0] = 0
        T[1] = 1
        for(i=2 to n) {
                T[i] = T[i-1]+T[i-2]
        }
        return T[n]
}
```



ตัวอย่างของฟังก์ชัน fibo

```
int fibo(int n) {
        a = 0
        b = 1
        while(n > 1) {
                t = a
                a = b
                b = b + t
                n--
        }
        return b
}
```

โดยปกติเราจะต้องเก็บค่า fibo(x) ของค่า x ทุกๆ ค่า ตั้งแต่ 0 ถึง n

เราจะปรับปรุงการใช้หน่วยความจำให้ดีขึ้นได้อย่างไร

เนื่องจากเราดูจากปัญหาแล้ว

fibo(n) = fibo(n-1) + fibo(n-2)

พบว่าเราไม่ได้ใช่ค่าเก่าอีก ทำให้เราไม่จำเป็นต้องเก็บค่าอื่นที่น้อยกว่านี้ไปได้

Running Time

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
&= T(n-2) + c + c \\
&= T(n-3) + c + c + c \\
&= \ldots \\
&= T(n-k) + ck \\
&= T(1) + (n-1)c \\
&= cn
\end{aligned}
$$

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

Steps ในการแก้ปัญหา Dynamic programming

- นิยาม subproblem
- หา recurrence ของ subproblem
    - เป็นการหาวิธีการรวมคำตอบของ subproblem ให้เป็นคำตอบที่ใหญ่ขึ้น
- แสดงและแก้ base case

Coin change:Dynamic programming

นิยาม subproblem

ให้ C[p] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

Coin change:Dynamic programming

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

Coin change:Dynamic programming

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 แล้วใช้ค่าต่ำสุด ให้ d_i แทนเหรียญที่ i

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

หา base case

Coin change:Dynamic programming

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 แล้วใช้ค่าต่ำสุด ให้ d_i แทนเหรียญที่ i

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

หา base case $C[0] = 0$

สมมติว่ามีแค่เหรียญ 1, 4, 5 และ 10

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

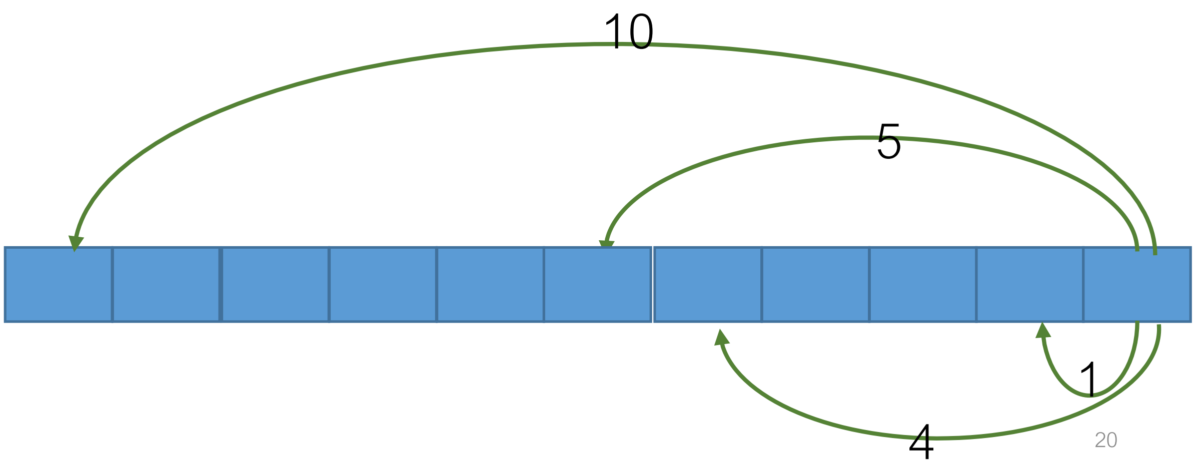
โครงสร้างของ pseudocode ในการแก้ปัญหา Dynamic programming

```
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

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

```
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;
}
```



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 (ถามจากค่าที่ดีสุดของ 0 บาท + 1)

2 บาทลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท (ถามจากค่าที่ดีสุดของ 1 บาท + 1)

3 บาทลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท (ถามจากค่าที่ดีสุดของ 2 บาท + 1)

4 บาทลองแทนว่าใช้เหรียญอะไรทอนได้

พบว่าใช้เหรียญบาท 4 เหรียญ (ถามจากค่าที่ดีสุดของ 3 บาท + 1)

กับ เหรียญ 4 บาท 1 เหรียญ (ถามจากค่าที่ดีสุดของ 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

กำหนดให้ n

Goal: ต้องการจำนวนวิธีที่ต่างกันในการเขียนค่า n โดยการใช้การรวมกันของตัวเลข 1, 3, 4

ตัวอย่าง

n = 5 มี 6 วิธี

5 = 1 + 1 + 1 + 1 + 1

= 1 + 1 + 3

= 1 + 3 + 1

= 3 + 1 + 1

= 1 + 4

= 4 + 1

นิยาม sub problem

ให้ D_n แทนจำนวนวิธีในการเขียน n เมื่อเป็นผลรวมของ 1, 3, 4

นิยาม sub problem

ให้ D_n แทนจำนวนวิธีในการเขียน n เมื่อเป็นผลรวมของ 1, 3, 4

หา recurrence

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

ถ้า x_m เป็น 1 ส่วนที่เหลือจะต้องรวมกันให้ได้ n-1

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

สำหรับกรณีอื่น (x_m=3, x_m=4) ก็คิดแบบเดียวกัน

พบว่า รูปแบบในการเขียน n เกิดได้จาก 3 รูปแบบนั่นคือ

ตัวสุดท้ายเขียนด้วย 1

จำนวนวิธีจะเท่ากับ D_{n-1}

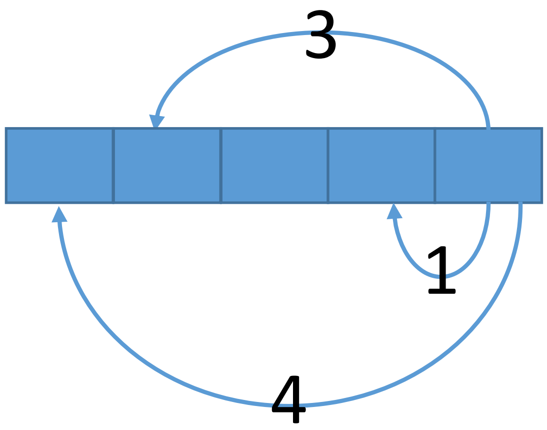
ตัวสุดท้ายเขียนด้วย 3

จำนวนวิธีจะเท่ากับ D_{n-3}

ตัวสุดท้ายเขียนด้วย 4

จำนวนวิธีจะเท่ากับ D_{n-4}

คำตอบได้จากนำทุกวิธีมารวมกันจะได้



หา recurrence

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

จัดการกรณี base case

$D_0 = 1$

$D_n = 0$ ถ้า n มีค่าน้อยกว่า 0

ครบหรือยัง(เทียบกับสมการเริ่มต้นว่าเริ่มทำงานได้หรือยัง)

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

```
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]
}
```

Running Time เป็นเท่าไร

Maximum Value Contiguous Subsequence

Maximum Value Contiguous Subsequence

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

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

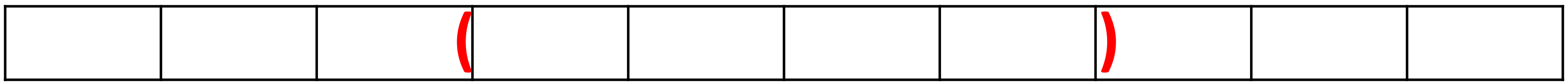
ตัวอย่างเช่น

1, 3, 5, -4, 2

จะได้ว่าเลือกลำดับ 1, 3, 5 ผลรวมมีค่าเป็น 9

1, -5, 2, -1, 3 มีผลรวมมากสุดเท่าไร?

Input: Array A[1..n] ของจำนวนจริง



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

นิยาม subproblem

ให้ B[j] แทนผลรวมที่ติดกันที่มากสุดเมื่อพิจารณาถึงช่องที่ j

เมื่อพิจารณาตัวที่ j

คำถาม: เมื่อพิจารณาตัวเลขตัวที่ j สิ่งที่เกิดขึ้นได้มีอะไรบ้าง

คำตอบ: นับรวมตัวที่ j หรือไม่รวมตัวที่ j

คำถาม: นับรวมตัวที่ j เพราะอะไร

คำตอบ: เมื่อรวมตัวที่ j แล้วทำให้ผลรวมมีค่ามากขึ้นมากกว่าเริ่มต้นนับใหม่ที่ตำแหน่งที่ j

คำถาม: ไม่รวมตัวที่ j เพราะอะไร

คำตอบ: เริ่มต้นนับใหม่ได้ผลรวมมากกว่า

ความสัมพันธ์

เขียน recurrence ได้

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

โดยที่

$\mathrm{B}[j] = B[j-1] + A[j]$ คือขยายช่วงของคำตอบมารวมช่องที่ j

$\mathrm{B}[j] = A[j]$ คือเริ่มต้นช่วงใหม่ที่มีแค่ $A[j]$

Base case

ถ้ามีตัวเดียวก็ต้องตอบเลย ไม่ว่าจะติดลบหรือไม่ก็ตาม

B[0] = A[0]

maxContiguousSum

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

maxContiguousSum(Update version)

```
int maxContiguousSum(int A[], int n)
{
        int max_so_far = A[0], i;
        int curr_max = A[0];
        for (i = 1; i < n; i++)
        {
                curr_max = max(A[i], curr_max+A[i]);
                max_so_far = max(max_so_far, curr_max);
        }
        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 ลำดับ A1, A2, … An

Goal: ต้องการหา subsequence (ส่วนของลำดับไม่จำเป็นต้องติดกัน) ที่เพิ่มขึ้นที่ยาวที่สุด

ตัวอย่างเช่น

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

ได้ subsequence ที่เพิ่มขึ้นยาว 5 ตัว

ได้แก่ 1, 3, 4, 6, 7

นิยาม sub problem

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

นิยาม sub problem

ให้ L(j) แทน subsequence ที่เพิ่มขึ้นที่ยาวที่สุด ณ ตัวที่ j

หา recurrence

พิจารณาตัวที่ j

คำถาม ตัวที่ j ควรจะนับต่อจากตัวไหน

คำตอบ ตัวที่มีค่าน้อยกว่ามันและมีผลการนับที่มากที่สุด

หา recurrence

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

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

Base case

L(1) = 1

อย่าลืม เราจะหาค่ามากสุด แต่ว่าตอนนี้เราดูที่ว่าถ้านับเอาตัวที่ j แล้วจะทำให้มากที่สุดอย่างไร ไม่อย่างนั้นต้องวนหาค่ามากสุดอีกครั้งด้วย

```
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
}
```