Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 หรือ 204252; และ 206183 หรือ 206281

ผู้สอน: ตอน 1 อ. เบญจมาศ ปัญญางาม

ตอน 2 อ. ดร. จักริน ชวชาติ

บทที่ 9

Dynamic Programming Part II

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 หรือ state
2. หาความสัมพันธ์เพื่อ หา recurrence ของ subproblem
   - เป็นการหาวิธีการรวมคำตอบของ subproblem ให้เป็นคำตอบที่ใหญ่ขึ้น
3. แสดงและแก้ base case

Coin change : Dynamic programming

1. นิยาม subproblem

ให้ C[p] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

Coin change : Dynamic programming

1. นิยาม subproblem

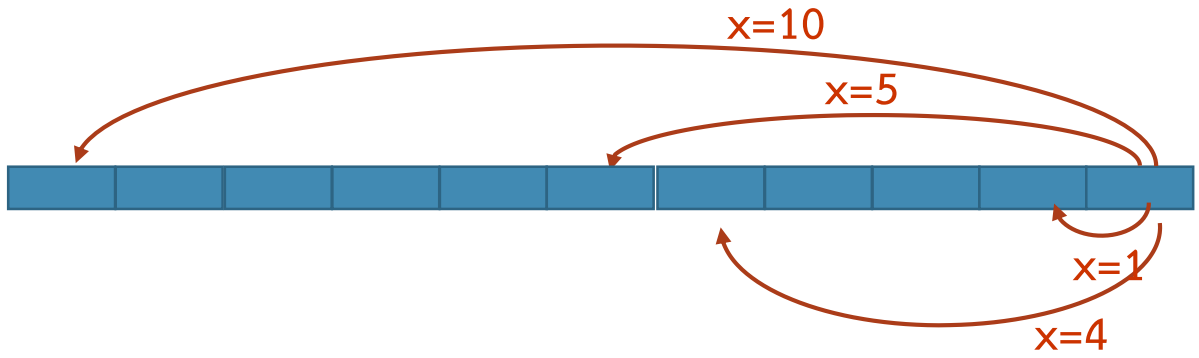
ให้ C[p] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

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 หรือ10 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

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

โดย ให้ d_i แทนเหรียญที่ i

3. หา base case คือ 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
}
```

Recursive

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

Topdown

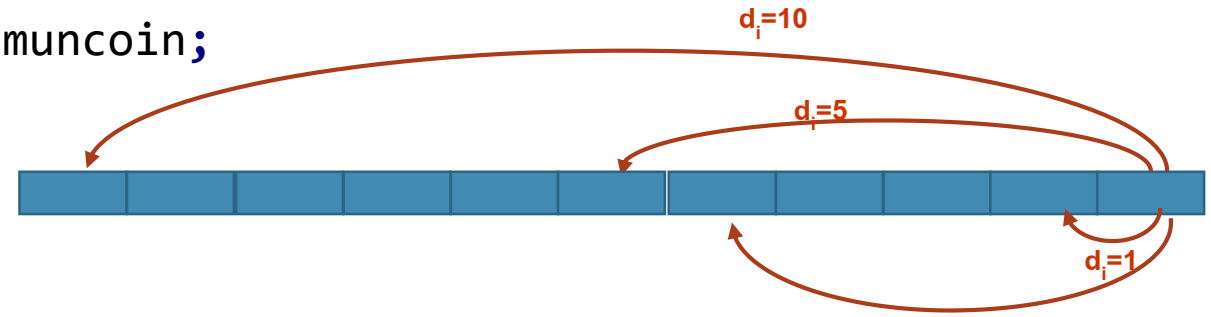
```
int d[4] = {1, 4, 5, 10}, k = 4;
int table[10000];
int CHANGE2(int n){
    if(n ==0){
        return 0;
    }
    if(n<0){
        return 0;
    }

    if(table[n]!=0)
        return table[n];

    int minimuncoin = n;
    for(int i =0;i<k;i++){
        if(n>=d[i])
            minimuncoin = min( minimuncoin, CHANGE2(n-d[i])+1 );
    }
    return table[n]=minimuncoin;
}
```

Bottomup

สมมติว่ามีแค่เหรียญ 1, 4, 5 และ 10

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

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



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 บาท ลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท

(ถามจากค่าที่ดีสุดในการแลกเงิน 0 บาท + 1)

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

ซึ่งจะได้คำตอบที่ดีที่สุดในการแลกเงิน 1 บาท เป็นจำนวน 1 เหรียญ (1+0)

2 บาท ลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท

(ถามจากค่าที่ดีสุดในการแลกเงิน 1 บาท + 1)

3 บาท ลองแทนว่าใช้เหรียญอะไรทอนได้ พบว่าใช้ได้เฉพาะเหรียญบาท

(ถามจากค่าที่ดีสุดในการแลกเงิน 2 บาท + 1)

4 บาท ลองแทนว่าใช้เหรียญอะไรทอนได้

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

Maximum Value Contiguous Subsequence

Maximum Value Contiguous Subsequence

กำหนดให้ ลำดับของเลขจำนวนจริง (มีทั้งเลขบวกและเลขลบ) n ตัว

$$A_1, A_2, ..., A_n$$

เป้าหมาย หาลำดับที่ติดกัน $A_i, ..., A_j$ ที่ผลรวมของสมาชิกในลำดับนั้นมีค่ามากที่สุด

ตัวอย่างเช่น

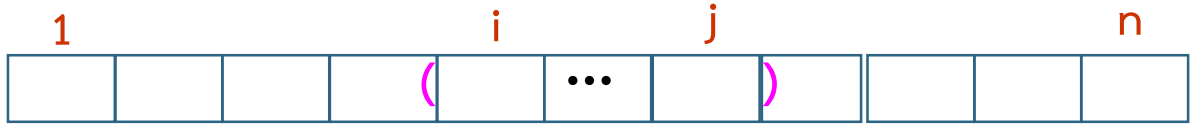
- กำหนดลำดับของเลขเป็น 1, 3, 5, -4, 2

  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

เนื่องจากสิ่งที่ต้องการคือ หาลำดับที่ติดกัน A_i, ..., A_j ที่ให้ผลรวมมากที่สุด

ดังนั้นพิจารณาตัวที่ j

คำถาม: เมื่อพิจารณาตัวเลขตัวที่ j สิ่งที่เกิดขึ้นได้มีอะไรบ้าง

คำตอบ: จะตัดสินใจเลือกรวมตัวที่ j หรือไม่รวมตัวที่ j

คำถาม: การเลือกรวมตัวที่ j เพราะเหตุผลอะไร

คำตอบ: เมื่อนำผลรวมนับรวมตัวที่ j แล้วทำให้ผลรวมมีค่ามากขึ้นโดยมากกว่า 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	ค่าA[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

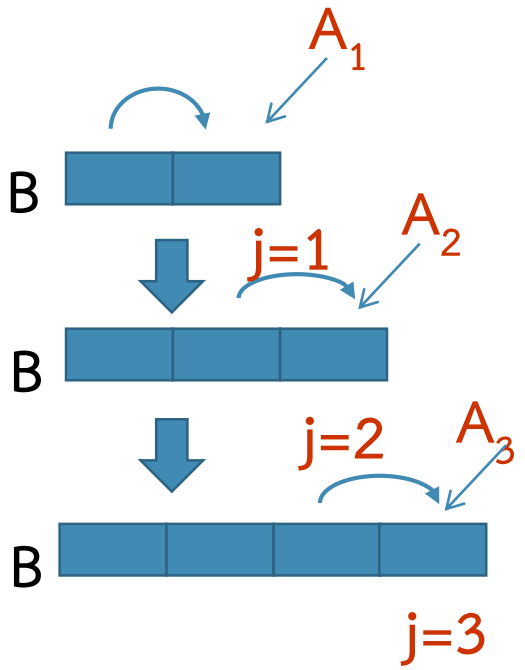
3. หา base case

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 $A_1, A_2, ..., A_n$

เป้าหมาย ต้องการหา subsequence (ส่วนของลำดับไม่จำเป็นต้องติดกัน)ที่เพิ่มขึ้นที่ยาวที่สุด

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	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
}
```