Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281

ผู้สอน: ตอน 1 อ. เบญจมาศ ปัญญางาม เรียน ห้อง 207
ตอน 2 อ. ดร. จักริน ชวชาติ เรียน ห้อง 209

วันสอบปลายภาค : วันศุกร์ที่ 13 พ.ย. 2563
เวลา 8:00 -11:00 น. (ตามประกาศมหาวิทยาลัย)

บทที่ 9

Dynamic Programming Part I

บทที่ 9
Dynamic Programming

Algorithmic Paradigms

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

Fibonacci Numbers

ก่อนที่จะใช้เทคนิคของ Dynamic Programming จะอธิบายถึงเทคนิคที่เกี่ยวข้องที่เรียกว่า Memorization 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

เริ่มต้นด้วย Algorithm แบบธรรมดาในการหา Fibonacci number ซึ่งจะเขียนตามนิยามของปัญหาการหา Fibonacci number ตัวที่ n

```
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

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

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

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

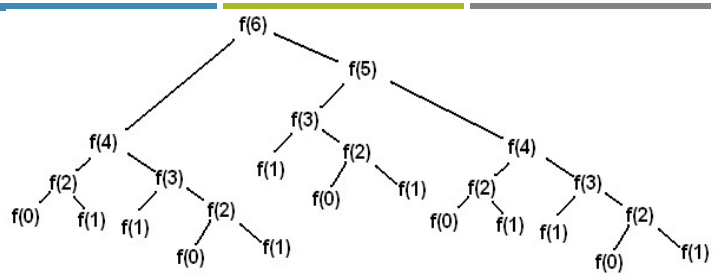
$\geq 8T(n-6) + 4c + 2c + c$

$\geq 2^kT(n-2k) + c(2^{k-1} + 2^{k-2} + ...+ 2^1 + 2^0)$

$\geq c2^k + c(2^k-1)/(2-1)$

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

ตัวอย่างการทำงาน



ข้อสังเกต

- มีการคำนวณซ้ำกันหลายจุดมาก
- โหนดใบ (Leaves) ของต้นไม้ (Tree) นี้จะเป็น 0 หรือ 1
- ผลรวมของค่าเหล่านี้จะถูกรวมขึ้นไปกลายเป็นผลลัพธ์สุดท้าย

Memorization Method

จากข้อสังเกตที่ว่า หลายๆ ปัญหาย่อยมีการคำนวณที่เหมือนกัน

- เพื่อลดเวลาในการคำนวณ เราจะคำนวณปัญหาย่อยเหล่านั้นแล้วเก็บผลลัพธ์ไว้ในตาราง เช่น array
- เมื่อต้องการแก้ปัญหาย่อยอีกครั้งเราจะนำเอาผลลัพธ์ที่คำนวณไว้แล้วตอบได้เลย

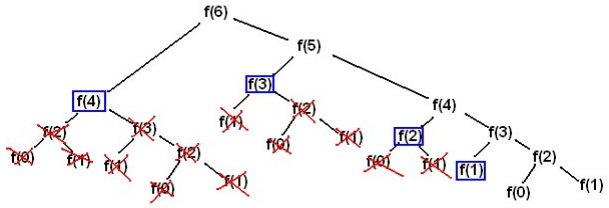
หลักการของ Memorization Method

1. การมองปัญหาแบบย้อนกลับ
2. การค้นหาผลลัพธ์ของปัญหาย่อยในตาราง
3. กรณีไม่พบผลลัพธ์ที่ต้องการในตาราง ให้คำนวณแบบ recursive ไปแล้วเก็บผลลัพธ์ที่ได้ไว้ในตาราง ก่อน return ค่า

Memorization Method

ข้อสังเกต เราสามารถเปลี่ยนอัลกอริทึมที่มีการ memorization แบบ recursive นี้มาเป็นอัลกอริทึมแบบทำซ้ำจากล่างขึ้นบน (bottom-up) ได้ โดยการเติมคำตอบของปัญหาย่อยลงในตาราง

ตัวอย่าง ปัญหาการหา Fibonacci number ตัวที่ n



หากค่าในกล่องสี่เหลี่ยมจะถูกคำนวณไว้แล้ว และถูกนำมาใช้

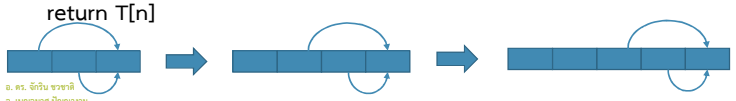
เห็นได้ชัดว่าอัลกอริทึมจะเร็วกว่าการคิดแบบตรงๆ (คำนวณทั้งหมด)

Memorization Method

ตัวอย่างการแก้ปัญหาการหา Fibonacci number ตัวที่ n

มีการคำนวณปัญหาย่อยเหล่านั้นแล้วเก็บผลลัพธ์ไว้ในตาราง T

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

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] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

Coin change : Dynamic programming

1. นิยาม subproblem

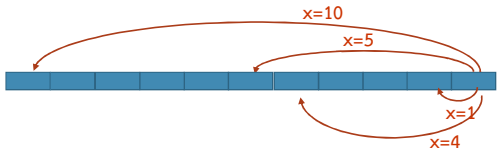
ให้ C[p] แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องการในการแลกเงิน p บาท

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

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

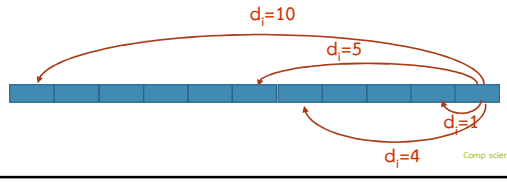
โครงสร้างของ pseudocode ในการแก้ปัญหา Dynamic programming

```
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

สมมติว่ามีแค่เหรียญ 1, 4, 5 และ 10
(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;
}
```



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

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

5 = 1 + 1 + 1 + 1 + 1

= 1 + 1 + 3

= 1 + 3 + 1

= 3 + 1 + 1

= 1 + 4

= 4 + 1

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 $n = x_1+x_2+...+ x_m$

ถ้า x_m เป็น 1 ส่วนที่เหลือจะต้องรวมกันให้ได้ n-1

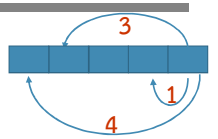
นั่นคือจำนวนของรูปแบบที่ตัวสุดท้ายที่เลือก

คือ $x_m = 1$ 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

$D_4 = D_3 + D_1 + D_0$

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

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

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

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