การโปรแกรมเชิงพลวัตร (1)

Dynamic Programming

อ. ลือพล พิพานเมฆาภรณ์

Content

- แนวคิดของการโปรแกรมเชิงพลวัตร (Dynamic Programming)
  - หลักการ top-down DP
  - หลักการ bottom-up DP
- ตัวอย่างโจทย์ปัญหาการ Search คำตอบโดยใช้ Dynamic Programming
  - Fibonacci number
  - Binomial Coefficient
  - Subset Sum

Concept of dynamic programming (DP)

- การโปรแกรมเชิงพลวัตร (Dynamic Programming) เป็นเทคนิคการแก้ปัญหาโดยมีแนวคิดในการแบ่งปัญหาออกเป็นปัญหาย่อย (sub-problems) คล้ายกับการแบ่งแยกและเอาชนะ แต่ใช้วิธีจำคำตอบ (memorization) เพื่อลดเวลาคำนวณปัญหาย่อยที่ซ้ำซ้อนกัน

- ดังนั้นประสิทธิภาพของ Dynamic Programming จึงมาจากเวลาที่ลดลงเนื่องจากไม่ต้องคำนวณคำตอบของปัญหาซ้ำซ้อนกัน
- ในการใช้งาน Dynamic Programming ต้องนิยามความสัมพันธ์ระหว่างคำตอบของปัญหาและคำตอบย่อยของมัน เรียกว่า **recurrence relation**
  - Dynamic Programming = recurrence relation + memorization

ตัวอย่าง Computing Fibonacci numbers

- ลำดับเลขอนุกรมไฟโบนักชี (Fibonacci numbers)

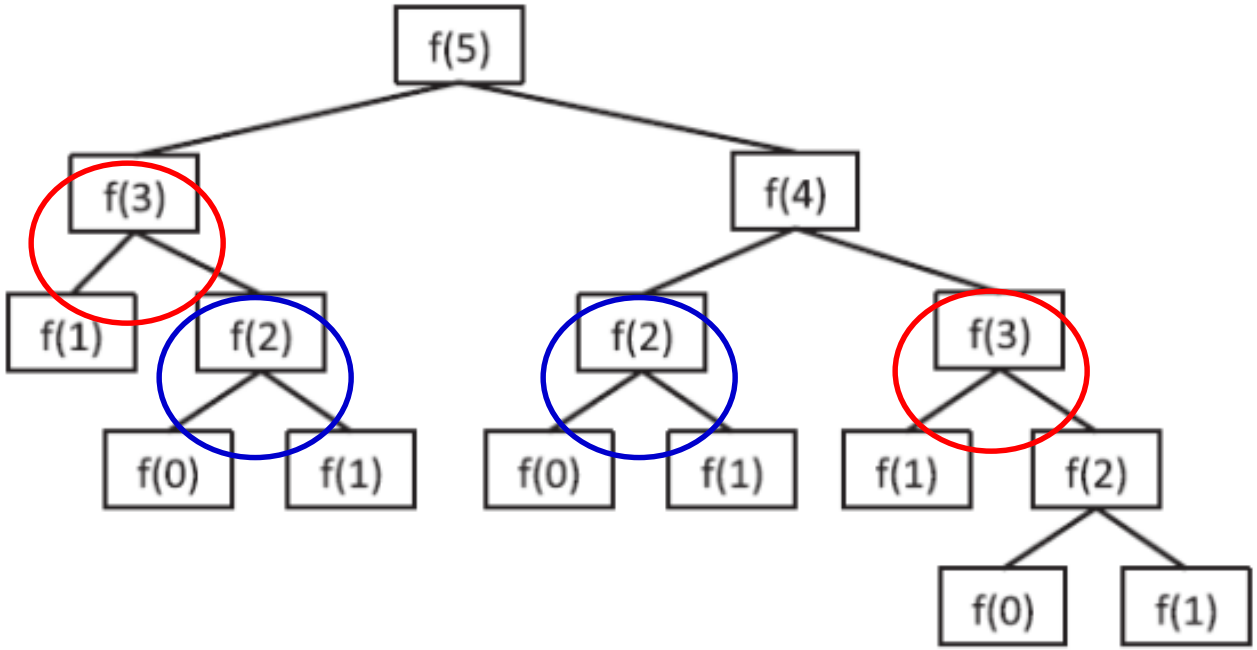
1, 1, 2, 3, 5, 8, 13, 21, ...

เราสามารถคำนวณหาค่าของพจน์ที่ n

$$fib(n) = \begin{cases} 1, & n = 0, 1 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

Recursive-based Fibonacci

```
int f(int n)
{ if(n < 2)
      return 1;
   return f(n-2)+f(n-1);
}
```



ต้นไม้รีเคอร์ชั่น

f(5)

Recursive-based Fibonanci

- เริ่มจาก recurrent relation fib(n) = f(n-1) + f(n-2)
- เตรียมหน่วยความจำ (memory) เพื่อบันทึกคำตอบย่อย
- ก่อนคำนวณคำตอบย่อย ให้ตรวจสอบว่ามีคำตอบอยู่ในหน่วยความจำหรือไม่

```
int memo[6]= {-1,-1,-1,-1,-1,-1};

int f_t(int n)
{ if (n < 2)
    return n;
  if(memo[n] != -1)
    return memo[n];

  memo[n] = f_t(n-1) + f_t(n-2);
  return memo[n];

}
```

ค่าเริ่มต้นของหน่วยความจำ

-1	-1	-1	-1	-1	-1

Recursive-based Fibonanci

- เริ่มจากการนิยาม recurrent relation fib(n) = f(n-1) + f(n-2)
- เตรียมหน่วยความจำ (memory) เพื่อบันทึกคำตอบย่อย
- ก่อนคำนวณคำตอบย่อย ให้ตรวจสอบว่ามีคำตอบอยู่ในหน่วยความจำหรือไม่

```
int memo[6]= {-1,-1,-1,-1,-1,-1};

int f_t(int n)
{ if (n < 2)
    return n;
  if(memo[n] != -1)
    return memo[n];

  memo[n] = f_t(n-1) + f_t(n-2);
  return memo[n];

}
```

```
-1  -1  1  2  3  5
```

Bottom-up Fibonanci

- เริ่มจากปัญหาย่อยที่เล็กที่สุด จากนั้นใช้ recurrence relation ที่นิยามเพื่อหาคำตอบของปัญหาที่ใหญ่ขึ้น

```
int f(int n) {
    memo[0]=0;
    memo[1]=1;
    for (int i = 2; i <= n; i++)
        memo[i] = memo[i-1] + memo[i-2];
    return memo[n];
}
```

0 1 1 2 3 5

ปัญหาที่เหมาะกับ Dynamic Programming

- ปัญหานั้นต้องสามารถนิยาม recurrence relation ได้

  เช่น f(n) = f(n-1) + f(n-2)

- คำตอบย่อยมีการทับซ้อนกัน (overlap sub-problems)

เช่น เมื่อเราคำนวณ F(4) = F(3) + F(2) และ F(3) = F(2) + F(1) จะเห็นว่า F(2) ถูกคำนวณซ้ำซ้อน

แบบฝึกหัด

จงตรวจสอบว่า สมการ recurrence relation ข้อใดเหมาะสำหรับการใช้ dynamic programming เพื่อหาคำตอบ

1. F(n, k) = F(n-1, k) + F(n-1, k-2) เมื่อ n >= k, F(n, 0) = 1, F(n, 1) = 1, F(n, n) = 1
2. F(n)=F(n−1)+n เมื่อ F(0)=0
3. F(n)=F(n−1)+F(n−2)+F(n−3) เมื่อ F(0)=0, F(1)=1, F(2)=2
4. F(n)=F(n−1) + F(n/2) เมื่อ F(0) = 1
5. F(n)=F(n−1) + 2*F(n−1) เมื่อ F(0) = 2

การเตรียมคำตอบของปัญหาย่อย (Memorization)

- เทคนิค dynamic programming มักจะใช้อาร์เรย์ (array) 1 มิติ หรือเมตริกซ์ (matrix) ที่ถูกออกแบบซึ่งสัมพันธ์กับสมการ recurrence relation ของปัญหา

- ดังนั้นเวลาที่ใช้ในการหาคำตอบของปัญหาก็คือการปรับปรุงค่า (update) ในตารางเมื่อได้คำตอบในแต่ละช่องภายในอาร์เรย์ดังกล่าว

- กระบวนการหาคำตอบมี 2 แบบ ดังนี้
  - แบบ top-down (Simple)
  - แบบ bottom-up (Efficiency improvement)

สัมประสิทธิ์ทวินาม (Binomial Coefficient)

- binomial coefficient **C(n, k)** หรือ $\binom{n}{k}$ คือจำนวนของเซตย่อยของข้อมูลขนาด k จำนวน จากเซตข้อมูลทั้งหมด n จำนวน ($0 \le k \le n$)

$$C(n, k) = \frac{n!}{(n-k)!*k!}$$

เช่น C(3,2) = 3! / (3-2)!*2! = 6 / 1*2 = 3

C(3, 0) = 3! / (3-0)!*0! = 1

C(3,3) = 3! / (3-3)!3! = 1

การแปลงเป็นสมการ recurrence relation

```
C(n, k) = n! / (k! * (n-k)!)
        = (n-1)! * n / (k! * (n-k)!)
        = (n-1)! * n / ((k-1)! * k * (n-k-2)! * (n-k-1) * (n-k))
        = (n-1)! / ((k-1)! * (n-k-1)!) * (n / k * (n-k))
        = [(n-1)! / ((k-1)! * (n-k-1)!)] * (1/(n-k) + 1/k)
        = (n-1)! / ((k-1)! * (n-k)!) + (n-1)! / (k! * (n-k-1)!)
        = C(n-1, k-1) + C(n-1, k)
```

$C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$ for $0 < k < n$

และ $C(n, 0) = 1$

$C(n, n) = 1$

ตัวอย่าง $C(3, 2) = C(2, 1) + C(2, 2) = 2 + 1 = 3$

$C(2, 1) = C(1, 0) + C(1 ,1)$

$= 1 + 1 = 2$

$C(2, 2) = 1$

แบบฝึกหัด

จงเขียนฟังก์ชั่นแบบ recursive เพื่อคำนวณ binomial coefficient C(n, k) จาก recurrence relation ต่อไปนี้

$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \quad \text{เมื่อ } 0 < k < n$$
$$C(n, 0) = 1$$
$$C(n, n) = 1$$

Dynamic Programming approach

- เตรียม table เพื่อเก็บคำตอบของ C(n, k) ในทุกค่าของ n และ k ที่เป็นไปได้
- เตรียมคำตอบของปัญหาที่เล็กที่สุดก่อน
- เลือกวิธีเติมคำตอบลงใน table (bottom-up หรือ top-down)

$C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$ for $0 < k < n$
$C(n, k) = 1$ k=0
$C(n, n) = 1$ n=k

สำหรับค่า n

สำหรับค่า k

	0	1	2	...	k-1	k
0	1					
1	1	1				
2	1	2	1			
⋮						
k	1					1
⋮						
n-1	1					
n	1					C(n, k)

การคำนวณ C(6, 4)

	0	1	2	3	4
0	1				
1	1	1			
2	1		1		
3	1			1	
4	1			C(4,3)	1
5	1			C(5,3)	C(5,4)
6	1				C(6,4)

$C(6, 4) = C(5, 3) + C(5, 4)$

เริ่มจาก

$C(6, 4) = C(5, 3) + C(5, 4)$

C(5, 3) = table[5, 3] หรือ หาคำตอบจาก
C(5, 3) = C(4, 2) + C(4, 3)

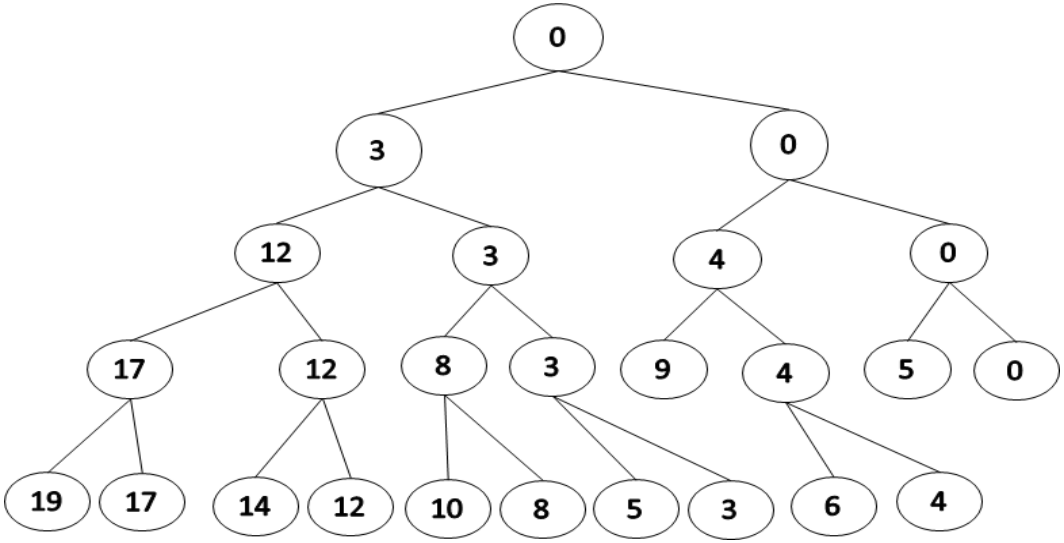
C(4, 2) = table[4, 2] หรือหาคำตอบจาก
C(4, 2) = C(3, 1) + C(3, 2)

Top-down DP for binomial coefficient

```
int dp[n][k];                                   // memory table

int C(int n, int k)
{
    if(k==0 || k==n)                            // base case
        return 1;

    if(dp[n][k] != -1)                          // existing compute
        return dp[n][k];

    dp[n][k]= C(n-1, k-1)+C(n-1, k);

    return dp[n][k];
}
```

Subset Sum

- ให้ A = {3, 4, 5, 2} ต้องการหาว่ามีเซตย่อยที่มีผลรวมเท่ากับ 6 หรือไม่
- จากตัวอย่างพบว่า {4, 2} จะมีผลรวมเท่ากับ 6
- โดยทั่วไปสามารถหาคำตอบด้วยวิธี backtracking อย่างไรก็ตามอาจใช้เวลามากหาก A มีขนาดใหญ่ เนื่องจากต้อง generate สถานะของคำตอบที่เป็นไปได้ทั้งหมด



- โดยหลักการของ Dynamic programming เราอาจนิยาม recurrence relation เพื่อหาคำตอบ Sum(n, k) โดยที่ n คือจำนวนสมาชิกใน A และ k คือค่าเป้าหมาย

Subset Sum

- ปัญหา Sum(n, k) อาจนิยามคำตอบ โดยพิจารณาจากสมาชิกตัวสุดท้าย A[n-1] ซึ่งแบ่งออกเป็น 2 กรณี

  กรณีที่ 1 หากคำตอบมี A[n-1] เป็นส่วนหนึ่งของคำตอบ ดังนั้น

  Sum(n, k) = Sum(n-1, k - A[n-1])

- A = {3, 4, 5, 2} และ k = 6

  Sum(4, 6) = Sum(3, 6-2) = Sum(3, 4)

  หากสมาชิก 3 ตัวหน้ามีผลรวมเท่ากับ 4 ก็แสดงว่ามีคำตอบ Sum(4, 6)

Subset of sum

- กรณีที่ 2 หากคำตอบไม่มี A[n-1] เป็นส่วนหนึ่งของคำตอบ ดังนั้น คำตอบก็อาจขึ้นอยู่ผลลัพธ์ของสมาชิกก่อนหน้า ซึ่งได้แก่

$$sum(n, k) = sum(n-1, k)$$

เช่น A = {3, 4, 5, 2} k = 9

Sum(4, 9) = Sum(3, 9) หมายถึง ถ้าคำตอบ Sum(3,9) เป็นจริง ดังนั้น Sum(4,9) ก็เป็นจริงด้วย

Sum (n, k) = Sum(n-1, k) or Sum(n-1, k - A[n-1])

การเตรียมคำตอบ Subset sum

k

A[n-1]/k	0	1	2	3	4	5	6
{}	T	F	F	F	F	F	F
{3}	T						
{3, 4}	T						
{3, 4, 5}	T				Sum(3, 4)		Sum(3, 6)
{3,4, 5, 2}	T						Sum(4, 6)

n

Sum(4, 6) = Sum(3, 6) or Sum(3, 6-2)

Sum (n, 0) = true

Sum (0, k) = false

Sum (n, k) = Sum(n-1, k) or Sum(n-1, k - A[n-1])

Sum(3, 6)

k

A[n-1]/k	0	1	2	3	4	5	6
{}	T	F	F	F	F	F	F
{3}	T						
{3, 4}	T	Sum(2,1)					Sum(2,6)
{3, 4, 5}	T				Sum(3, 4)		Sum(3, 6)
{3,4, 5, 2}	T						Sum(4, 6)

n

Sum(3, 6) = Sum(2, 6) or Sum(2, 6-5)

Sum (n, 0) = true

Sum (0, k) = false

Sum (n, k) = Sum(n-1, k) or Sum(n-1, k - A[n-1])

Sum(2, 6)

k

A[n-1]/k	0	1	2	3	4	5	6
{}	T	F	F	F	F	F	F
{3}	T		Sum(1,2)				Sum(1,6)
{3, 4}	T	Sum(2,1)					Sum(2,6)
{3, 4, 5}	T				Sum(3, 4)		Sum(3, 6)
{3,4, 5, 2}	T						Sum(4, 6)

n

Sum(2, 6) = Sum(1, 6) or Sum(1, 6-4)

Sum (n, 0) = true

Sum (0, k) = false

Sum (n, k) = Sum(n-1, k) or Sum(n-1, k - A[n-1])

Sum(1, 6)

k

A[n-1]/k	0	1	2	3	4	5	6
{}	T	F	F	F	F	F	F
{3}	T		Sum(1,2)				Sum(1,6)
{3, 4}	T	Sum(2,1)					Sum(2,6)
{3, 4, 5}	T				Sum(3, 4)		Sum(3, 6)
{3,4, 5, 2}	T						Sum(4, 6)

n

Sum(1, 6) = Sum(0, 6) or Sum(0, 6-3)

= F or F = F

Sum (n, 0) = true

Sum (0, k) = false

Sum (n, k) = Sum(n-1, k) or Sum(n-1, k - A[n-1])

Bottom-up Subset Sum

A[n-1]/k	0	1	2	3	4	5	6
{}	T	F	F	F	F	F	F
{3}	T	F	F	T			
{3, 4}	T						
{3, 4, 5}	T						
{3,4,5,2}	T						

Sum(1, 1) = Sum(0, 1) or Sum(0, 1 - 3) = F
Sum(1, 2)= Sum(0, 2) or Sum(0, 2 - 3) = F
Sum(1,3) = Sum(0, 3) or Sum(0, 3 - 3) = T

Sum (n, 0) = true
Sum (0, k) = false
Sum (n, k) = Sum(n-1, k) or Sum(n-1, k - A[n-1])

Subset of sum

A[n-1]/k	0	1	2	3	4	5	6
{}	T	F	F	F	F	F	F
{3}	T	F	F	T	F	F	F
{3, 4}	T	F	F	T	T	F	F
{3, 4, 5}	T						
{3,4,5,2}	T						

Sum(2, 3) = Sum(1, 3) or Sum(1, 3-4) = T
Sum(2, 4)= Sum(1, 4) or Sum(1, 4-4) = T
Sum(2, 5) = Sum(1, 5) or Sum(1, 5-4) = F
Sum(2, 6) = Sum(1,6) or Sum(1, 6-4) = F

Subset of sum

A[n-1]/k	0	1	2	3	4	5	6
{}	T	F	F	F	F	F	F
{3}	T	F	F	T	F	F	F
{3, 4}	T	F	F	T	T	F	F
{3, 4, 5}	T	F	F	T	T	T	F
{3,4,5,2}	T	F	T	T	T	T	T

Sum(4, 6) = Sum(3, 6) or Sum(3, 6-2) = T

Dynamic Programming

```
int Sum(int n ,int k)
{ bool S[n+1][W+1];
    for (int i=0; i<=n; i++)        // initial Sum(n,0)
            S[i][0] = true;
    for (int j=1; j<=K; j++)         // initial Sum(0, k)
            S[0][j] = false;

     for (int j=1; j<=K; j++) {
        for (int i=1; i<=n; i++){
            if (j < A[i])                    // out-of-table
                S[i][j] = S[i-1][j];
            else
                S[i][j] = S[i-1][j-1] || S[i-1][j-A[i]];
           }
      }
  return S[n][k];
}
```