บทที่ 3

กำหนดการพลวัต (Dynamic Programming)

จากบทที่ 2 จะเป็นอัลกอริทึมในลักษณะ top-down ที่แบ่งเป็นปัญหาย่อยๆ แล้วหาคำตอบจากปัญหาเล็กๆ แล้วรวบรวมเป็นคำตอบของปัญหา อย่างกรณี merge sort ที่แต่ละคำตอบเล็กๆ ไม่ซ้ำซ้อนกัน เพราะ แต่ละอาเรย์ย่อยต่างก็แยกกันจัดเรียงลำดับ โดยไม่ต้องงเกี่ยวข้องกับสมาชิกของอาเรย์ย่อยอื่น ในทางตรงข้ามกับการแก้ปัญหา Fibonacci พจน์ที่ n ต้องหาค่าของพจน์ที่ n-1 กับ พจน์ที่ n-2 ทำให้ต้องมีการเรียกซ้ำแล้วต้องเรียกซ้ำอีก เป็นผลให้ห้ค่า complexity โตแบบ exponential แต่ถ้าเราเก็บผลลัพธ์ของพจน์ก่อนหน้าเพื่อใช้ในการประมวลผลพจน์ถัดไป โดยเก็บผลลัพธ์ที่ได้จากการประมวลผลในแต่ละขั้นตอนไว้ในอาเรย์เพื่อสามารถนำไปใช้ในภายหลัง ซึ่งเทคนิคหนึ่งที่ใช้แนวคิดนี้ก็คือ กำหนดการพลวัต

กำหนดการพลวัตคล้ายคลึงกับอัลกอริทึมการแบ่งแยกกับการเอาชนะตรงที่มีการแบ่งแยกปัญหาใหญ่ออกเป็นปัญหาย่อยที่มีลักษณะเช่นเดิม แต่ที่ต่างกันคือ กำหนดการพลวัตมีการเก็บผลลัพธ์ย่อย ๆ ไว้ใช้ในการหาผลลัพธ์ของปัญหาที่ใหญ่ขึ้น แทนการคำนวณซ้ำๆ โดยทั่วไปกำหนดการพลวัต การทำงานจะเป็นในลักษณะจากล่างขึ้นบน (bottom-up) โดยมีขั้นตอนในการแก้ปัญหาโดยกำหนดการพลวัตมีดังนี้

1. หาความสัมพันธ์ซ้ำที่จะใช้ในการแก้ปัญหา
2. แก้ปัญหาจากล่างขึ้นบน โดยแก้ปัญหาย่อยก่อน และเก็บผลลัพธ์จากปัญหาย่อยใส่ลงในอาเรย์เพื่อใช้ในการแก้ปัญหาที่ใหญ่ขึ้น

ในบทนี้จะใช้กำหนดการพลวัตในการแก้ปัญหาสัมประสิทธิทวินาม(Binomial coefficient), การคูณห่วงโซ่เมตริกซ์(matrix chain multiplication) และการหาลำดับร่วมที่ยาวที่สุด (longest common subsequence)

3.1 สัมประสิทธิทวินาม (Binomial Coefficient)

Binomial Theorem :

$$(a+b)^n = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} a^k b^{n-k}$$

$$\text{Binomial coefficient} = \binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad for\ 0 \le k \le n$$

สำหรับค่า n, k โตๆ การคำนวณค่า n!, k! จะเกิด overflow ทำให้ไม่สามารถหาค่าสัมประสิทธิจากสมการนี้ได้ แต่เราสามารถปรับสมการใหม่เป็น

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \; or \; k = n \end{cases}$$

ตัวอย่าง 3.1 จงหาค่าสัมประสิทธิของ $(a+b)^5$

โปรแกรมหาค่าสัมประสิทธิทวินามด้วยวิธีการแบ่งแยกและเอาชนะ

(Binomial Coefficient Using Divide and conquer)

```
//โปรแกรม Binomial1.java
public class Binomial1
  {  public static int binCoef(int n, int k)
     {
         if(k == 0 || n ==k)
          {  return 1;  }
         else
          {return binCoef(n-1, k-1)+binCoef(n-1, k);}
       }// binomial Coefficient
       public static void main(String[] args)
       {
            int i,j,c ; c=0;
            for (i =0;i<= 5;++i )
            {
               for (j=0; j<=i;++j )
               {
                    c=binCoef(i,j);
                    System.out.print(c+"\t");
                } // end for j
                System.out.println();
             } //end for i
         }//main
}//class Binomial1
```

Outputs

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

Complexity ของกรณีนี้จะเหมือนกับกรณีของ Fibonacci คือ ประมาณ $O(2^n)$

โปรแกรมหาค่าสัมประสิทธิทวินามด้วยวิธีกำหนดการพลวัต

(Binomial Coefficient Using Dynamic programming)

1. ตั้ง recursive case :

$$c[i][j] = \begin{cases} c[i-1][j-1] + c[i-1][j] & 0 < j < i \\ 1 & j = 0 \text{ or } j = i \end{cases}$$

2. แก้ปัญหาย่อยๆ ในรูปแบบ bottom-up โดยเริ่มจาก c[0][0], c[1][0], c[1][1] เมื่อถึง c[2][1] ก็จะสามารถนำค่าในอาเรย์ c[1][0]+c[1][1] โดยไม่ต้องคำนวณค่า c[1][0] และc[1][1] ซ้ำอีก ตามโปรแกรม Binomial2.java

Complexity ของกรณีนี้ คือ $\sum_{i=0}^{n}\sum_{j=0}^{j=i}1$

$$\sum_{i=0}^{n}\sum_{j=0}^{i}1 = \sum_{i=0}^{n} i - 0 + 1$$

$$= \sum_{i=0}^{n} i + 1$$

$$= (\sum_{i=0}^{n} i) + n + 1$$

$$= \frac{n(n+1)}{2} + n + 1$$

$$= \frac{n^2 + 3n + 2}{2}$$

$O(n^2)$

```
 public class Binomial2
{
 static int b[][] = new int[20][20];
 public static int binCoef2(int n, int k)
 {
  if(k == 0 || n ==k)
   {b[n][k]=1;}
  else
   {b[n][k]=b[n-1][k-1]+b[n-1][k];}
  return b[n][k];
 }//
 public static void main(String[] args)
  {
  int i,j,c ; c=0;
  for (i =0;i<= 5;++i )
  {
   for (j=0; j<=i;++j )
   {
   c=binCoef2(i,j);
   System.out.print(c+"\t");
   }
  System.out.println();
  }
  }//main
 } // binomial Coefficient2
```

3.2 การคูณห่วงโซ่เมตริกซ์ (matrix chain multiplication)

ถ้า matrix C = matrix A ขนาด 2x3 คูณกับ matrix B ขนาด 3x4 จะได้ matric มีขนาดเท่ากับ 2x4 โดยที่สมาชิกของ matrix C ตัวที่ i,j มีค่าเท่ากับ

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

สมาชิกตัวที่ c_{11} จะมีค่าเท่ากับ $c_{11} = a_{11}\, b_{11} + a_{12}\, b_{21} + a_{13}\, b_{31}$

สมาชิก matrix c 1 ตัวจะมีการคูณกันทั้งหมด 3 ครั้ง matrix c มีสมาชิกทั้งหมด 8 จำนวน จะมีการคูณทั้งหมด = 2x3x4 = 24 ครั้ง โดยจะเท่ากับ จำนวนแถวของ A x จำนวนแถวของ B x จำนวนหลักของ B

A B C=AxB

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \text{x} \begin{bmatrix} 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 \\ 5 & 7 & 8 & 2 \end{bmatrix} = \begin{bmatrix} 19 & 28 & 34 & 19 \\ 43 & 64 & 79 & 52 \end{bmatrix}$$

คราวนี้มาดูว่าถ้ามีเมตริกซ์ n เมตริกซ์ คือ $\{ A_1, A_2, A_3, .., A_n \}$ จะกำหนดลำดับการคูณเพื่อทำให้การคูณเมตริกซ์ทั้ง n เมตริกซ์มีจำนวนครั้งของการคูณเกิดขึ้นน้อยที่สุดอย่างไร

ตัวอย่าง การคูณเมตริกซ์จำนวน 4 เมตริกซ์ $\{A_1, A_2, A_3, A_4\}$ มีขนาดของเมตริกซ์เป็นเมตริกซ์ P จะได้

A_1 A_2 A_3 A_4

(20x2) (2x30) (30x10) (10x8)

$P=\{ p_0, p_1, p_2, p_3\} = \{20, 2, 30, 10, 8\}$

เราสามารถใส่วงเล็บเพื่อกำหนดลำดับการคูณได้ 5 แบบ คือ

1. $A_1 (A_2(A_3 A_4))$ 30x10x8 + 2x30x8 + 20x2x8 = 3,200
2. $(A_1 A_2)(A_3 A_4)$ 20x2x30 + 30x10x8 + 20x30x8 = 8,400
3. $A_1 ((A_2A_3)A_4)$ 2x30x10 + 2x10x8 + 20x2x8 = 1,080
4. $((A_1A_2)A_3)A_4$ 20x2x30 + 20x30x10 + 20x10x8 = 8,800
5. $(A_1 (A_2A_3))A_4$ 2x30x10 + 20x2x10 + 20x10x8 = 2,600

จะได้ว่าลำดับการคูณแบบที่ 3 $A_1 ((A_2A_3) A_4)$ ใช้จำนวนครั้งในการคูณน้อยที่สุด หากจำนวนเมตริกซ์มากๆ และมีขนาดใหญ่มากๆ การใส่วงเล็บเพื่อกำหนดลำดับการคูณจึงมีความสำคัญมาก

3.2.1 วิธีกำหนดการพลวัต (Dynamic Programming Approach)

เราจะใช้กำหนดการพลวัตในการคูณห่วงโซ่เมตริกซ์ โดยมีขั้นตอน ดังนี้

1. หาคำตอบในรูปความสัมพันธ์ซ้ำ (recursive solution) โดยพิจารณาจำนวนครั้งในการคูณในการหาผลคูณเมตริกซ์ $A_i \ldots A_j$

$$m[i][j] = \begin{cases} 0 & if\ i = j \\ \min\limits_{i \le k \prec j} \{m[i][k] + m[k+1][j] + p_{i-1} p_k p_j\} & if\ i < j \end{cases}$$

โดยที่ m[i,j] = จำนวนครั้งของการคูณที่น้อยที่สุดในการหาผลคูณเมตริกซ์ $A_i \ldots A_j$

k แทนตำแหน่งที่ใส่วงเล็บ

ในกรณี i = j คือมีเมตริกซ์เดียว ไม่มีการคูณกัน m[I,j] = 0 สำหรับกรณีที่มีการคูณเมตริกซ์มากกว่า 2 เมตริกซ์ จะต้องเลือก m[i,j] ที่มีค่าน้อยที่สุด และเพื่อที่จะได้ตามรอยการใส่วงเล็บ เราจะเก็บค่า k ไว้ในอาเรย์ s[i, j]

2. เมื่อเก็บผลลัพธ์ของปัญหาย่อยๆไว้แล้ว ก็จะรวบรวมเพื่อเป็นคำตอบของปัญหาหลัก

<u>ตัวอย่าง 3.2</u> จงใส่วงเล็บเพื่อกำหนดลำดับการคูณของเมตริกซ์ $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ โดยที่

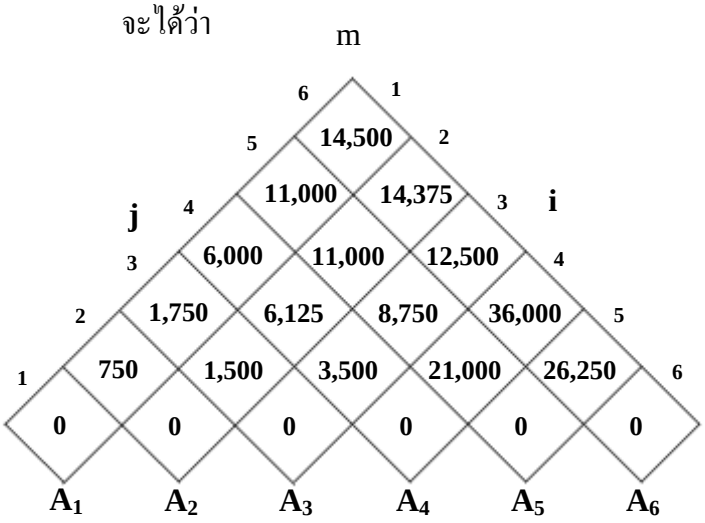
เมตริกซ์	ขนาด
A_1	30 x 40
A_2	40 x 15
A_3	15 x 35
A_4	35 x 10
A_5	10 x 20
A_6	20 x 25

$\{p_0, p_1, p_2, p_3, p_4, p_5, p_6\} = \{10, 15, 5, 20, 35, 30, 25\}$

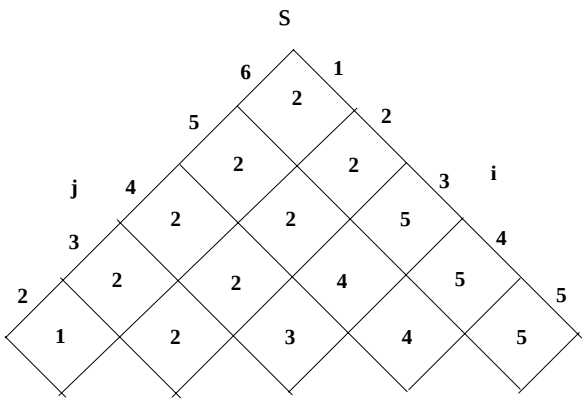
วิธีทำ จาก

$$m[i][j] = \begin{cases} 0 & if\ i = j \\ \min\limits_{i \le k \prec j} \{m[i][k] + m[k+1][j] + p_{i-1} p_k p_j\} & if\ i < j \end{cases}$$

จะได้ว่า



รูป 3.1 (ก) ตารางเก็บจำนวนครั้งของการคูณที่ดีที่สุดของผลคูณเมตริกซ์ $A_i \ldots A_j$



รูป 3.1 (ข) ตารางเก็บค่า k ทำให้ได้จำนวนครั้งของการคูณที่ดีที่สุด

โปรแกรม3.3 MatrixChain.java

```
//โปรแกรม MatrixChain.java
class MatrixChain
 {    static  int  m[][] = new int[51][51]; //  กำหนดจำนวนเมตริกซ์ไม่เกิน
       static  int   s[][] = new int[51][51];
       public static void  MatrixChainOrder(int[] p)
      { int  i,j,q,k,h;
        int  n =  6;            // n คือ จำนวนเมตริกซ์ที่ต้องการหาผลคูณตามตัวอย่างนี้
        // กำหนดค่าเริ่มต้น  m[i, i]  =  0  และ  s[i,ij] = 0 เพราะเป็น เมทริกซ์เดียวไม่มีการคูณ
       for( i = 1; i <= n;  i++)
          {  m[i][i] = 0;
             s[i][i]  = 0;   }
       for( h = 2; h <= n; h++)
          {    for(  i =  1;  i <= n-h+1; i++)
                 {   j = i + h -1;
                   m[i][j] =1000000  ;     //  กำหนดให้มีค่ามาก ๆ แทนค่า ∞
                   for ( k =  i; k <=  j-1;  k++)
                      {  q = m[i][k] + m[k+1][j] +p[i-1]*p[k]*p[j];
                         System.out.print(i+"\t"+j+"\t"+k+"\tm["+i+"]["+k+"]"+"+
                                  m["+(k+1)+"]["+j+"]"+" + p"+(i-1)+"p"+k+"p"+j);
                         System.out.print("\t="+m[i][k]+"+"+m[k+1][j]+"+"
                                  +( p[i-1]*p[k]*+p[j])+"\t"+q);
                        if (  q < m[i][j]  )
                          {   m[i][ j] = q;
                              s[i][j] = k;  }//if
                        System.out.println();
```

```
                                }//next k
                              }//next i
                            }//next h
          }// end MatrixChainOrder
           public  static void  Printorder  ( int  i,  int j)
          {   int k;
              if  (  i == j  )
                  System.out.print("A"+ i);
              else  {
                      k =  s[i][ j];
                      System.out.print("(");
                      Printorder( i,  k);
                      Printorder( k+1,  j);
                      System.out.print( ")" );  } //end else
          }// end PrintOrder
//-------------------------------------------------------------
   public static void main (String args[])
    {  int matrixSize [] = {10, 15, 5, 20, 35 ,30, 25};
       MatrixChainOrder(matrixSize);
       int i,j,h,k;
      for (h=6;h>=1;--h )
         { for(k=1;k<=h+1;++k)
            {  System.out.print("\t");  }
           for(k=1;k<=7-h;++k)
            {   i=k;j=k+h-1;
               System.out.print(m[i][ j ]+"\t\t");
             }//next k
           System.out.println( );
        }//next h
      for (h=6;h>=2;--h )
         {  for(k=2;k<=h+1;++k)
              {System.out.print("\t");}
              for(k=1;k<=7-h;++k)
                 {   i=k;j=k+h-1;
                    System.out.print(s[i][ j ]+"\t\t");
                 }//next k
             System.out.println( );
        }//next h
        Printorder(1,6);
}//main
}//class MatrixChain
```

i	j	k	m[i][j]			s[i][j]

1	2	1	m[1][1]+ m[2][2] + p0p1p2	0+0+750	750	1
2	3	2	m[2][2]+ m[3][3] + p1p2p3	0+0+1500	1,500	2
3	4	3	m[3][3]+ m[4][4] + p2p3p4	0+0+3500	3,500	3
4	5	4	m[4][4]+ m[5][5] + p3p4p5	0+0+21000	21,000	4
5	6	5	m[5][5]+ m[6][6] + p4p5p6	0+0+26250	26,250	5
1	3	1	m[1][1]+ m[2][3] + p0p1p3	0+1500+3000	4,500	
1	3	2	m[1][2]+ m[3][3] + p0p2p3	750+0+1000	1,750	2
2	4	2	m[2][2]+ m[3][4] + p1p2p4	0+3500+2625	6,125	2
2	4	3	m[2][3]+ m[4][4] + p1p3p4	1500+0+10500	12,000	
3	5	3	m[3][3]+ m[4][5] + p2p3p5	0+21000+3000	24,000	
3	5	4	m[3][4]+ m[5][5] + p2p4p5	3500+0+5250	8,750	4
4	6	4	m[4][4]+ m[5][6] + p3p4p6	0+26250+17500	43,750	
4	6	5	m[4][5]+ m[6][6] + p3p5p6	21000+0+15000	36,000	5
1	4	1	m[1][1]+ m[2][4] + p0p1p4	0+6125+5250	11,375	
1	4	2	m[1][2]+ m[3][4] + p0p2p4	750+3500+1750	6,000	2
1	4	3	m[1][3]+ m[4][4] + p0p3p4	1750+0+7000	8,750	
2	5	2	m[2][2]+ m[3][5] + p1p2p5	0+8750+2250	11,000	2
2	5	3	m[2][3]+ m[4][5] + p1p3p5	1500+21000+9000	31,500	
2	5	4	m[2][4]+ m[5][5] + p1p4p5	6125+0+15750	21,875	
3	6	3	m[3][3]+ m[4][6] + p2p3p6	0+36000+2500	38,500	
3	6	4	m[3][4]+ m[5][6] + p2p4p6	3500+26250+4375	34,125	
3	6	5	m[3][5]+ m[6][6] + p2p5p6	8750+0+3750	12,500	5
1	5	1	m[1][1]+ m[2][5] + p0p1p5	0+11000+4500	15,500	
1	5	2	m[1][2]+ m[3][5] + p0p2p5	750+8750+1500	11,000	2
1	5	3	m[1][3]+ m[4][5] + p0p3p5	1750+21000+6000	28,750	
1	5	4	m[1][4]+ m[5][5] + p0p4p5	6000+0+10500	16,500	
2	6	2	m[2][2]+ m[3][6] + p1p2p6	0+12500+1875	14,375	2
2	6	3	m[2][3]+ m[4][6] + p1p3p6	1500+36000+7500	45,000	
2	6	4	m[2][4]+ m[5][6] + p1p4p6	6125+26250+13125	45,500	
2	6	5	m[2][5]+ m[6][6] + p1p5p6	11000+0+11250	22,250	
1	6	1	m[1][1]+ m[2][6] + p0p1p6	0+14375+3750	18,125	
1	6	2	m[1][2]+ m[3][6] + p0p2p6	750+12500+1250	14,500	2
1	6	3	m[1][3]+ m[4][6] + p0p3p6	1750+36000+5000	42,750	
1	6	4	m[1][4]+ m[5][6] + p0p4p6	6000+26250+8750	41,000	
1	6	5	m[1][5]+ m[6][6] + p0p5p6	11000+0+7500	18,500	

การใส่วงเล็บของการคูณเมตริกซ์ $\{A_1, A_2, A_3, A_4, A_5, A_6\}$

1. ดูได้จากตารางที่ $i = 1,\ j = 6$ จะได้ $k = 2$ นั่นคือ $(A_1A_2)\,(A_3\,A_4\,A_5\,A_6)$

2. หาการใส่วงเล็บของการคูณเมตริกซ์ $\{A_3, A_4, A_5, A_6\}$ จากตารางที่ $i = 3,\ j = 6$ จะได้ $k = 5$ นั่นคือ $(A_1A_2)((A_3\,A_4\,A_5)\,A_6)$

3. หาการใส่วงเล็บของการคูณเมตริกซ์ $\{A_3, A_4, A_5\}$ จากตารางที่ $i = 3,\ j = 5$ จะได้ $k = 4$ นั่นคือ $((A_1A_2)(((A_3\,A_4\,)A_5)\,A_6))$

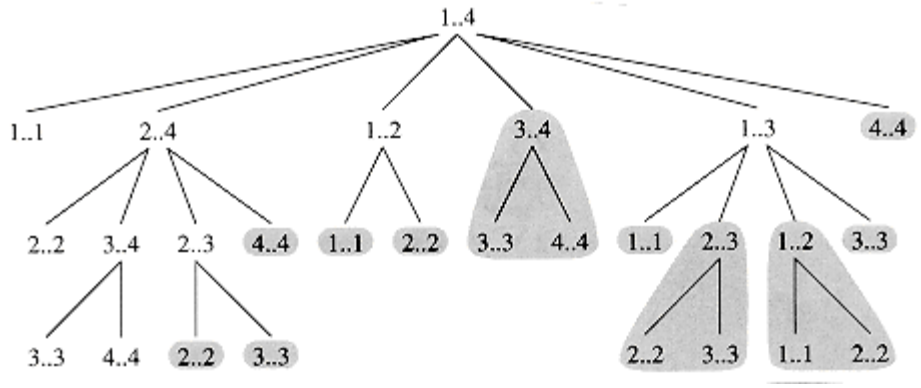
ดังนั้น การคูณเมตริกซ์ $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ จะมีลำดับการคูณเป็น $((A_1A_2)(((A_3\,A_4\,)A_5)\,A_6))$ และจำนวนครั้งของการคูณน้อยที่สุดเท่ากับ 14,500 ครั้ง

ความซับซ้อนของอัลกอริทึม

จากเมท็อด MatrixChainMult ที่ใช้กำหนดการพลวัตในการหาจำนวนครั้งที่น้อยที่สุดที่ใช้ในการคูณเมตริกซ์ $A_i,\ldots, A_j$ จะพบว่ามีจำนวนครั้งของการคูณและการกำหนดค่าเป็น $O(n^3)$

3.2.2 วิธีการแบ่งแยกและเอาชนะ (Divide and conquer Approach)

เราสามารถจะใช้วิธีเรียกซ้ำ(recursive) เพื่อแก้ปัญหานี้ได้เช่นเดียวกัน การใช้อัลกอริทึมแบบเรียกซ้ำในการคำนวณ จะพบว่าแต่ละปัญหาย่อยจะมีการแตกกิ่งก้านทำการเรียกตนเองอีกหลายครั้งแทนการใช้ค่าที่ได้จากการคำนวณก่อนหน้านั้น การหาค่า m[3][4] ต้องมีการคำนวณค่า m[3][3] กับ m[4][4] เช่นเดียวกับการคำนวณค่า m[2][4] ตามรูป 3.2 สังเกตุส่วนที่แรเงา



รูป 3.2 วิธีการเรียกซ้ำจะเรียกตนเองหลายครั้ง จากส่วนที่แรเงา

โปรแกรม 3.4 RecursiveMatrixChain

```
      static  int  m[][] = new int[51][51];
      static  int   s[][] = new int[51][51];
public static void  RecursiveMatrixChain(int[] p, int i, int j)
{int q;
  if(i==j)
  {return;}
  else
  {  m[i][j] = 1000000;
     for (k=i;k<=j-1 ;++k )
    { q= RecursiveMatrixChain(p,i,k) +  RecursiveMatrixChain(p,k+1,j) + p[i-1]*p[k]*p[j];  }
     if (q<m[i][j])
       {  m[i][j] = q; }
     return m[i][j];
   }
}// end method RecursiveMatrixChain
```

Complexity

แสดงความสัมพันธ์แบบ recursive

$T(1) \geq 1,$

$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \quad for\ n > 1$

$T(n) \geq 1 + T(1) + T(n-1) + T(2) + T(n-2) + \ldots + T(n-1) + T(1) + n - 1$

$T(n) \geq 2\sum_{i=1}^{n-1} T(i) + n$

$T(1) \geq 1 = 2^0$

$T(n) \geq 2\sum_{i=1}^{n-1} 2^{i-1} + n$

$= 2\sum_{i=1}^{n-2} 2^i + n$

$= 2(2^{n-1} - 1) + n$

$= 2^n - 2 + n$

$O(2^n)$

เนื่องจาก $\sum_{i=0}^{n-1} 2^i = 2^n - 1$

ทำให้มีจำนวนครั้งของการคูณและการกำหนดค่าเป็น $O(2^n)$

3.3 การหาลำดับร่วมที่ยาวที่สุด (longest common subsequence)

ในทางชีววิทยา เรามักต้องการเปรียบเทียบ DNA ว่าเป็นบุคคลคนเดียวกัน, เป็นญาติกัน หรือเป็นเผ่าพันธ์เดียวกันหรือไม่ โดยวิเคราะห์จาก DNA ที่ประกอบด้วยโมเลกุลของ adenine(A), guanine(G), cytosine(C) และ thymine (T) ตัวอย่างเช่น

S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

และ S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

ต้องการหาว่า DNA S_1 กับ S_2 มีความคล้ายกันมากน้อยอย่างไร ถ้าเหมือนกันก็จะเป็น DNA จากคนเดียวกัน ถ้าคล้ายกันมากก็มีโอกาสเป็นพี่น้องกัน วิธีหนึ่งที่ใช้วัดระดับความคล้ายก็คือ หาว่ามีลำดับของ A, C, G, T ที่ร่วมกันของ S_1 กับ S_2 กี่จำนวน (common subsequence)

ลำดับร่วม (subsequence) ของลำดับใด ๆ ก็คือ ลำดับเดิมที่ตัดสมาชิกบางตัวทิ้งไป กำหนดให้ $X = \{x_1, x_2, ..., x_m\}$ และ $Z = \{z_1, z_2, ..., z_k\}$ จะกล่าวว่า Z เป็นลำดับร่วมของ X หากมีลำดับของดัชนีของ X เป็น $\{i_1, i_2, ..., i_k\}$ โดยที่ $i_1 < i_2 < ... < i_k$

สำหรับทุกค่าของ j (j =1,..., k) จะมี $x_i = z_j$ เช่น $Z = \{D, C, B, A\}$ เป็นลำดับร่วมของ $X = \{A, B, D, A, C, B, D, A\}$ โดยมีลำดับของดัชนีของ X เป็น { 3, 5, 6, 8 }

หากมีลำดับ 2 ลำดับ คือ X, Y จะกล่าวว่า Z เป็นลำดับร่วมของ X, Y ถ้า Z เป็นลำดับร่วมของทั้งลำดับ X และ Y เช่น $X = \{A, B, C, B, D, A, B\}$ และ $Y = \{B, D, C, A, B, A\}$ จะได้ $Z = \{B, C, A\}$ เป็นลำดับร่วมลำดับหนึ่งของ X, Y แต่ $Z = \{B, C, A\}$ ไม่ใช่ลำดับร่วมที่ยาวที่สุด เนื่องจาก Z มีขนาด 3 ตัวอักขระ $V = \{B, D, A, B\}$ และ $W = \{B, C, B, A\}$ เป็นลำดับร่วมของ X, Y ซึ่งมีขนาด 4 ตัวอักขระ เนื่องจากไม่มีลำดับร่วมขนาด 5 ตัวอักขระหรือมากกว่า ดังนั้น $V = \{B, D, A, B\}$ และ $W = \{B, C, B, A\}$ จึงเป็นลำดับร่วมที่ยาวที่สุด (longest common subsequence :LCS) ของลำดับ X, Y

ในการแก้ปัญหาลำดับร่วมที่ยาวที่สุดแบบถึกๆ (brute-force) ก็คือการสร้างทุกลำดับย่อย(subset) ที่เป็นไปได้ ของ X และ Y แล้วดูว่ามีลำดับย่อย Xi ใดตรงกับลำดับย่อย Yi และมีจำนวนลำดับยาวที่สุด ด้วยวิธีการถึกๆแบบนี้ จะต้องสร้างลำดับย่อยของ X จำนวนถึง 2^m เมื่อ m คือจำนวนสมาชิกของลำดับ X นั่นคือจะมี complexity เป็น exponential ซึ่งไม่เหมาะกับลำดับที่มีจำนวนสมาชิกมากๆ

เราจะใช้กำหนดการพลวัตในการหาลำดับร่วมที่ยาวที่สุด โดยมีขั้นตอน ดังนี้

1. นิยามลำดับร่วมที่ยาวที่สุดของลำดับ X, Y ในรูปความสัมพันธ์ซ้ำ ดังนี้

$$c[i][j] = \begin{cases} 0 & if\ i=0\ or\ j=0 \\ c[i-1][j-1]+1 & if\ i,j>0\ and\ x_i = y_j \\ \max(c[i][j-1], c[i-1][j] & if\ i,j>0\ and\ x_i \neq y_j \end{cases}$$

c[i][j] คือ ความยาวของลำดับร่วมที่ยาวที่สุด เมื่อลำดับ X มีความยาว i ตัวอักขระและลำดับ Y มีความยาว j ตัวอักขระ

2. ทำการเก็บผลลัพธ์ของปัญหาย่อย โดยในที่นี้จะทำการเก็บค่า c[i][j] และทิศทางที่ชี้ไปยังลำดับร่วมที่ยาวที่สุดใส่ลงในตาราง

โปรแกรม 3.5

```
public class LongestCS
{
  static int c[][] = new int[51][51];
  static String b[][] = new String[51][51];
  static String s1[] = {"","1","0", "1", "1", "0", "1", "1", "0"};
  static String s2[] = {"","0", "1", "0", "1", "1", "0", "1", "1", "0"};
  static int m =  8;
  static int n  =  9;
    public  static void LCS(String x[], String y[] )
     {  int i,j;
          for (  i = 0;  i <= m;  i++)    {c[i][0] = 0;b[i][0] ="\t\t"; }
          for (  j = 0;  j <= n;  j++)   { c[0][j] = 0;b[0][j]="";}
     for (  i = 1;  i <= m;  i++)
          {     for (  j = 1;  j <= n;  j++)
               {    if (  x[i] .equals(y[j]))
                     {    c[i][j] = c[i-1][j-1]+1;
                          b[i][j] = "\\";             }//if
                       else  if (  c[i-1][ j]  >=  c[i][ j-1]  )
                             {       c[i][j] =c[i-1][j];
                                     b[i][j] =  "|";          }//else if
                               else   {
                                          c[i][j] = c[i][j-1];
                                          b[i][j] =  "-";
                                       }//else
                  }//next j
     }//next i
     } //LCS
public  static void printLCS(String d[][], String x[], int i, int j )
     {
       if  ( i  > 0 &&  j > 0 )
        {   if  ( d[i][j] .equals( "\\"))
            {    printLCS(d,x, i-1, j-1);
                 System.out.print(x[i]);
             }
             else  if  ( d[i][j] .equals( "|") )
                   {     printLCS(d,x, i-1, j);             }
```

```
                    else   printLCS(d,x, i, j-1);
   }
  }// print-LCS
//-------------------------------------------------------
  public static void main(String[] args)
   {   int i,j;
       LCS(s1,s2);
       System.out.print("\tj\t");
       for(j=0;j<=n;++j){System.out.print(j+"\t");}
       System.out.println();
      System.out.print("i\t\tYj\t");
      for(j=1;j<=n;++j){ System.out.print(s2[j]+"\t");}
      System.out.println();
      for(i= 0;i<=m;++i)
        {
           if(i==0){System.out.print("Xi\t");}
            for (j=0;j<=n ;++j )
              { System.out.print(b[i][j]+"\t");}
           System.out.println();
           System.out.print(i+"\t"+s1[i]+"\t");
           for (j=0;j<=n ;++j )
          { System.out.print(c[i][j]+"\t"); }
          System.out.println();
        }  //end for i
    printLCS(b,s1,m,n);
    System.out.println();
  }//main
}  // end LongestCS
```

ตัวอย่าง 3.3 จงหาลำดับร่วมที่ยาวที่สุดของ X, Y โดยที่กำหนดให้

$X = \{1, 0, 1, 1, 0, 1, 1, 0\}$ และ

$Y = \{0, 1, 0, 1, 1, 0, 1, 1, 0\}$

วิธีทำ

ให้ $c[i][j] = 0$ เมื่อ $i = 0$ หรือ $j = 0$ เริ่ม

1. พิจารณา c[1][1] เนื่องจาก $x_1 = 0$ และ $y_1 = 1$ เกิดกรณี $x_i \neq y_j$ จะได้ c[1][1] มีค่าเท่ากับค่าที่มากที่สุดระหว่าง c[1][0] และ c[0][1] คือ c[1][1] = 0 และเนื่องจาก c[1][0] = c[0][1] (ในกรณีที่เท่ากัน) เราจะเลือก b[1][1] มีทิศเป็น "|"

2. พิจารณา c[1][2] เนื่องจาก $x_1 = 0$ และ $y_2 = 0$ เกิดกรณี $x_i = y_j$ ค่า c[1][2] = c[0][1]+1 = 0+1 = 1 และ b[1][2] มีทิศเป็น "\"

3. พิจารณา c[1][3] เนื่องจาก $x_1 = 0$ และ $y_3 = 1$ เกิดกรณี $x_i = y_j$ ค่า c[1][3] = ค่าที่มากที่สุดระหว่าง c[1][2] = 1 และ c[0][3] = 0 จะได้ c[1][3] = 1 และ b[1][3] มีทิศเป็น "-"

จากนั้นพิจารณาต่อไปเรื่อย ๆ จนครบทุกค่า i และ j ดังแสดงในตาราง

ตาราง แสดงการคำนวณค่า LCS ของลำดับ X, Y

	j	0	1	2	3	4	5	6	7	8	9
i		Yj	0	1	0	1	1	0	1	1	0
0	Xi	0	0	0	0	0	0	0	0	0	0
1	1	0	| 0	\ 1	- 1	\ 1	\ 1	- 1	\ 1	\ 1	- 1
2	0	0	\ 1	| 1	\ 2	- 2	- 2	\ 2	- 2	- 2	\ 2
3	1	0	| 1	\ 2	| 2	\ 3	\ 3	- 3	\ 3	\ 3	- 3
4	1	0	| 1	\ 2	| 2	\ 3	\ 4	- 4	\ 4	\ 4	- 4
5	0	0	\ 1	| 2	\ 3	| 3	| 4	\ 5	- 5	- 5	\ 5
6	1	0	| 1	\ 2	| 3	\ 4	\ 4	| 5	\ 6	\ 6	- 6
7	1	0	| 1	\ 2	| 3	\ 4	\ 5	| 5	\ 6	\ 7	- 7
8	0	0	\ 1	| 2	\ 3	| 4	| 5	\ 6	| 6	| 7	\ 8

จะได้ค่า LCS = 8 (ค่า c[8][9]) และลำดับร่วมจะหาได้จากการท่องตารางจากช่อง i = 8 และ j = 9 ไปตามทิศของ b[i][j] ถ้าเป็นทิศ "\" เราจะได้ว่า x[i][j] ตำแหน่งนั้นเป็นลำดับร่วม แล้วท่องตารางไปจนกระทั่งถึงตำแหน่ง b[i][1] ดังต่อไปนี้

1. b[8][9] เป็น "\" ให้ท่องไปตามแนวทแยง คือ b[7][8]
2. b[7][8] เป็น "\" ให้ท่องตามแนวทแยง คือ b[6][7]
3. b[6][7] เป็น "\" ให้ท่องตามแนวทแยง คือ b[5][6]
4. ท่องไปเรื่อย ๆ ตามทิศทางจนพบช่อง i = 0 และ j = 0 ซึ่งช่องผลลัพธ์ที่ได้จะแสดงด้วยการระบายสี
5. สุดท้ายจะได้ลำดับร่วมเป็น 10110110

ตัวอย่าง 3.4 จงหาลำดับร่วมที่ยาวที่สุดของ S_1, S_2 โดยที่กำหนดให้ S_1 = {"A", "B", "C", "B", "D", "A", "B"} และ S_2 ={"B","D","C","A","B","A"};

i	j	0	1	2	3	4	5	6
		Yj	B	D	C	A	B	A
0	Xi	0	0	0	0	0	0	0
1	A	0	| 0	| 0	| 0	\ 1	- 1	\ 1
2	B	0	\ 1	- 1	- 1	| 1	\ 2	- 2
3	C	0	| 1	| 1	\ 2	- 2	| 2	| 2
4	B	0	\ 1	| 1	| 2	| 2	\ 3	- 3
5	D	0	| 1	\ 2	| 2	| 2	| 3	| 3
6	A	0	| 1	| 2	| 2	\ 3	| 3	\ 4
7	B	0	\ 1	| 2	| 2	| 3	\ 4	| 4

จะได้ลำดับร่วมยาวที่สุด คือ **"BCBA"**

ตัวอย่าง 3.5 จงหาลำดับร่วมที่ยาวที่สุดของ S_1, S_2 โดยที่กำหนดให้

S_1 =
{"A","C","C","G","G","T","C","G","A","G","T","G","C","G","C","G","G","A","A","G","C","C","G","G","C","C","G","A","A"};

S_2 =
{"G","T","C","G","T","T","C","G","G","A","A","T","G","C","C","G","T","T","G","C","T","C","T","G","T","A","A","A"};

จะได้ลำดับร่วมยาวที่สุด คือ **"GTCGTCGGAAGCCGGCCGAA"**

ด้วยวิธีกำหนดการพลวัต ในการหาลำดับร่วมที่ยาวที่สุด ใน method LCS จะได้ complexity เป็น O(mn) และ method printLCS จะมี complexity = O(m+n) เพราะแต่ละ i, j จะลดลงในแต่ละรอบของการเรียกซ้ำ

แบบฝึกหัด

1. จงใส่วงเล็บเพื่อกำหนดลำดับการคูณของเมตริกซ์ $\{A_1, A_2, A_3 A_4, A_5, A_6\}$ ให้มีจำนวนครั้งของการคูณน้อยที่สุด โดยกำหนดให้ A_1 : 2x50 A_2 : 50x40 A_3 : 40x10 A_4 : 10x20 A_5 : 20x100 A_6 : 100x10

2. จงหาลำดับร่วมที่ยาวที่สุดของ X, Y โดยที่
กำหนดให้ $X = \{A, B, D, C, D, A, B, D, A\}$ และ $Y = \{B, C, A, D, B, A, B\}$