บทที่ 4
ขั้นตอนวิธีละโมบ (Greedy Algorithms)

ขั้นตอนวิธีละโมบจะทำงานเป็นขั้นตอน โดยจะตัดสินใจเลือกทางเลือกที่ดีที่สุดในขั้นตอนนั้น ๆ โดยไม่คำนึงถึงการตัดสินใจของขั้นตอนในอดีตและอนาคต โดยหวังว่าการตัดสินใจที่ดีที่สุดในแต่ละขั้นตอนจะนำไปสู่การตัดสินใจที่ดีที่สุดของปัญหาโดยรวม ขั้นตอนวิธีละโมบมักให้ผลลัพธ์ที่ดี ง่าย และใช้เวลาในการแก้ปัญหาไม่นานนัก โดยผลลัพธ์ที่ได้ไม่จำเป็นต้องเป็นผลลัพธ์ที่ดีที่สุด โดยทั่วไปขั้นตอนวิธีละโมบมักนิยมใช้แก้ปัญหา Optimization

ในกรณีที่ผลลัพธ์ที่ต้องการไม่จำเป็นต้องเป็นผลลัพธ์ที่ดีที่สุด ขั้นตอนวิธีละโมบมักถูกนำมาใช้ในการหาผลลัพธ์ที่ดีและง่าย แทนการหาผลลัพธ์ที่ดีที่สุดแต่ซับซ้อนและใช้เวลาในการหาผลลัพธ์นาน

เมื่อเทียบกับกำหนดการพลวัตที่จะใช้วิธีการแบ่งปัญหาเป็นปัญหาย่อยๆ สำหรับขั้นตอนวิธีละโมบ จะไม่มีการแบ่งเป็นปัญหาย่อย แต่จะได้คำตอบโดยการลำดับทางเลือกที่เหมือนจะเป็นทางเลือกดีที่สุด ที่คาดว่าจะเป็นคำตอบ (global optimal) หรืออาจจะเป็น local optimal ก็ได้

มีปัญหาจริงมากมายที่เป็นตัวอย่างการใช้ขั้นตอนวิธีละโมบ ตัวอย่างแรกได้แก่ การทอนเงินให้มีจำนวนธนบัตรและเหรียญน้อยที่สุด หากคุณต้องทอนเงิน 127 บาท คุณจะทอนด้วยธนบัตร 100 บาท 1 ใบ ธนบัตร 20 บาท 1 ใบ เหรียญ 5 บาท 1 เหรียญ และเหรียญ 1 บาท 2 เหรียญ ซึ่งได้จากการพิจารณาทอนธนบัตรและเหรียญที่มีมูลค่ามากก่อน

ขั้นตอนวิธีละโมบ มักเกี่ยวข้องกับการเรียงลำดับข้อมูลจากมากไปน้อย หรือน้อยไปมาก ตัวอย่างการใช้ขั้นตอนวิธีละโมบในปัญหาการจัดลำดับอย่างง่าย (simple scheduling problem) ปัญหาการเลือกกิจกรรม (activity selection problem) ปัญหาการจัดลำดับงาน (Task scheduling problem) ปัญหาถุงเป้ (knapsack problem) และการบีดอัดข้อมูลโดยใช้ Huffman Code (data compression by Huffman Code)

4.1 การจัดลำดับอย่างง่าย (simple scheduling problem)

กำหนดให้ งาน $j_1, j_2, j_3, ..., j_n$ ซึ่งมีเวลาที่ใช้ในการทำงานเป็น $t_1, t_2, t_3,..., t_n$ ตามลำดับ ในกรณีที่มีเพียง 1 หน่วยประมวลผล เราจะจัดลำดับงานอย่างไรเพื่อให้งานเหล่านี้มีเวลาดำเนินการแล้วเสร็จโดยเฉลี่ยน้อยที่สุด ในส่วนนี้เราจะกำหนดให้เป็น nonpreemptive scheduling นั่นคือ เมื่อเริ่มทำงานใดแล้วจะทำงานนั้นไปเรื่อย ๆ จนแล้วเสร็จ

การจัดลำดับงานที่มีเวลาแล้วเสร็จโดยเฉลี่ยน้อยที่สุด ทำได้โดยการจัดลำดับงานตามเวลาที่ใช้ในการทำงานจากน้อยไปมาก

ตัวอย่าง 4.1 กำหนดให้ มีงานทั้งหมด 3 งาน คือ j_1, j_2, j_3 ซึ่งมีเวลาที่ใช้ในการทำงานเป็น 5, 10, 4 นาที ตามลำดับ จงจัดลำดับงานเพื่อให้มีเวลาแล้วเสร็จโดยเฉลี่ยน้อยที่สุด
คำนวณเวลาที่ต้องใช้ทั้งหมด

งาน(job)	เวลาที่ต้องใช้
1	5 (นาทีในการทำงาน)
2	5 (นาทีคอย) + 10 (นาทีในการทำงาน)
3	15 (นาทีคอย) + 4 (นาทีในการทำงาน)
	รวม 39 นาที เฉลี่ย 13 นาที/งาน

ลองจัดลำดับงานในทุกรูปแบบ

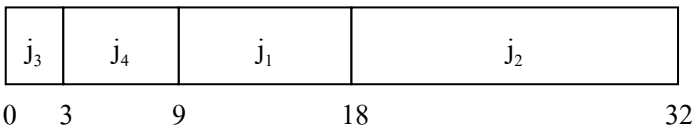
ลำดับงาน	เวลาที่ต้องใช้	รวม	เฉลี่ย
1, 2, 3	5 + (5+10) + (5+10+4)	39	13.00
1, 3, 2	5 + (5+4) + (5+4+10)	33	11.00
2, 1, 3	10 + (10 + 5) + (10+5+4)	44	14.67
2, 3, 1	10 + (10+4) + (10+4+5)	43	14.33
3, 1, 2	4 + (4+5) + (4+5+10)	32	10.67
3, 2, 1	4 + (4+10) + (4+10+5)	37	12.33
ลำดับงาน 3, 1, 2 ดีที่สุด รวมเวลา = 32 นาที เฉลี่ย = 10.67นาที			

ถ้าพิจารณาทุกลำดับงานเวลาที่ใช้คำนวณ(complexity) จะเป็น O(n!) เมื่อ n เป็นจำนวนงานทั้งหมด แต่ถ้าเราเรียงลำดับเวลาที่ทำงานจากน้อยไปมาก คือ งานที่ 3, 1, 2 จากนั้นเลือกลำดับงาน 3, 2, 1 เป็นคำตอบ วิธีนี้ complexity จะขึ้นกับการเลือกใช้วิธีการเรียงลำดับ ถ้าใช้ merge sort ก็จะได้ complexity = O(n log n)

กำหนดให้ มีงานทั้งหมด 4 งาน คือ j_1, j_2, j_3, j_4 ซึ่งมีเวลาที่ใช้ในการทำงานเป็น 9, 14, 3, 6 ตามลำดับ จงจัดลำดับงานเพื่อให้มีเวลาแล้วเสร็จโดยเฉลี่ยน้อยที่สุด

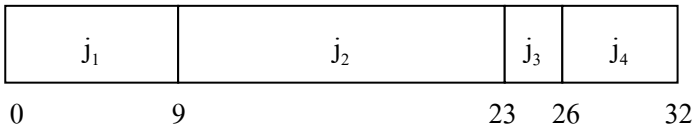
วิธีทำ

เรียงลำดับงานที่ใช้เวลาในการทำงานจากน้อยไปมาก ซึ่งได้ลำดับเป็น j_3, j_4, j_1, j_2 และมีเวลาแล้วเสร็จโดยเฉลี่ยเป็น (3+9+18+32)/4 = 15.5



รูปที่ 4.1 แสดงการจัดลำดับงานโดยขั้นตอนวิธีละโมบ

หากเรียงลำดับแบบเข้าก่อนออกก่อน(FIFO) จะได้ลำดับงานเป็น j_1, j_2, j_3, j_4 และมีเวลาแล้วเสร็จโดยเฉลี่ยเป็น (9+23+26+32)/4 = 22.5



รูป 4.2 แสดงการจัดลำดับงานแบบเข้าก่อนออกก่อน

จะเห็นว่า การจัดลำดับอย่างง่ายโดยขั้นตอนวิธีละโมบให้เวลาแล้วเสร็จโดยเฉลี่ยน้อยกว่า

4.2 กรณีที่มีหน่วยประมวลผลหลายหน่วย

ในกรณีที่มีหน่วยประมวลผลหลายหน่วย เราสามารถจัดลำดับงานที่มีเวลาแล้วเสร็จโดยเฉลี่ยน้อยที่สุดได้ โดยทำการจัดลำดับงานให้แก่แต่ละหน่วยประมวลผลตามเวลาที่ใช้ในการทำงานจากน้อยไปมาก โดยเรียงจากหน่วยประมวลผลที่ 1, 2, ..., n และวนกลับมาที่หน่วยประมวลผลที่ 1, 2, ..., n ซ้ำไปเรื่อย ๆ จนครบทุกงาน

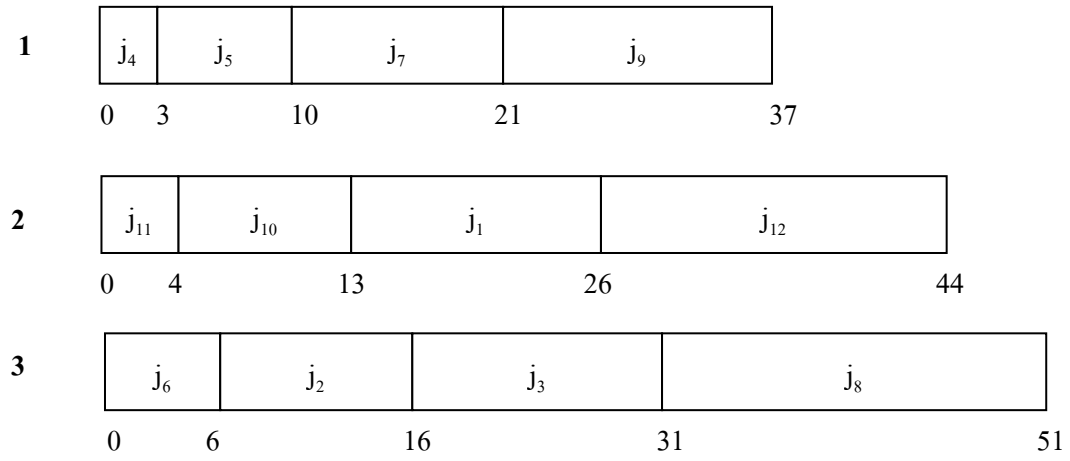
ตัวอย่าง 4.2 กำหนดให้ มีหน่วยประมวลผล 3 หน่วย และมีงานทั้งหมด 12 งาน คือ $j_1, j_2, j_3, j_4, j_5, j_6, j_7, j_8, j_9, j_{10}, j_{11}, j_{12}$ ซึ่งมีเวลาที่ใช้ในการทำงานเป็น 13, 10, 15, 3, 6, 7, 11, 20, 16, 9, 4, 18 ตามลำดับ จงจัดลำดับงานเพื่อให้มีเวลาแล้วเสร็จโดยเฉลี่ยน้อยที่สุด

วิธีทำ

เรียงลำดับงานที่ใช้เวลาในการทำงานจากน้อยไปมาก ซึ่งได้ลำดับเป็น

งาน	4	11	6	5	10	2	7	1	3	9	12	8
เวลา	3	4	6	7	9	10	11	13	15	16	18	20

หน่วยประมวลผลที่



รูป 4.3 แสดงการจัดลำดับงานโดยขั้นตอนวิธีละโมบ

กรณีที่ต้องการเวลาแล้วเสร็จโดยรวมน้อยที่สุด

ในกรณีที่มีหน่วยประมวลผลหลายหน่วย เราสามารถจัดลำดับงานที่มีเวลาแล้วเสร็จโดยรวมน้อยที่สุดได้ โดยทำการจัดลำดับงานให้แก่แต่ละหน่วยประมวลผลตามเวลาที่ใช้ในการทำงานจากน้อยไปมาก โดยเรียงจากหน่วยประมวลผลที่ 1, 2, ..., n และวนกลับมาที่หน่วยประมวลผลที่ n, n-1, n-2, ..., 1 ซ้ำไปเรื่อย ๆ จนครบทุกงาน

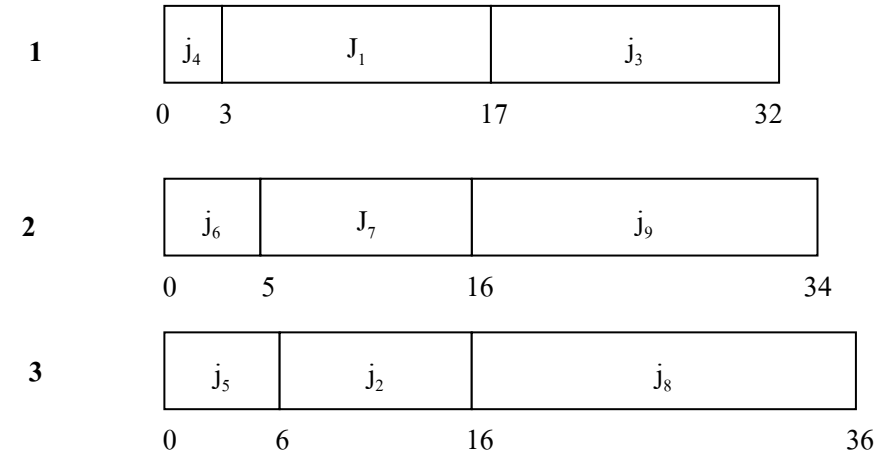
ตัวอย่าง 4.3 กำหนดให้ มีหน่วยประมวลผล 3 หน่วย และมีงานทั้งหมด 9 งาน คือ $j_1, j_2, j_3, j_4, j_5, j_6, j_7, j_8, j_9$ ซึ่งมีเวลาที่ใช้ในการทำงานเป็น 14, 10, 15, 3, 6, 5, 11, 20, 18 ตามลำดับ จงจัดลำดับงานเพื่อให้มีเวลาแล้วเสร็จโดยรวมน้อยที่สุด

วิธีทำ

เรียงลำดับงานที่ใช้เวลาในการทำงานจากน้อยไปมาก ซึ่งได้ลำดับเป็น

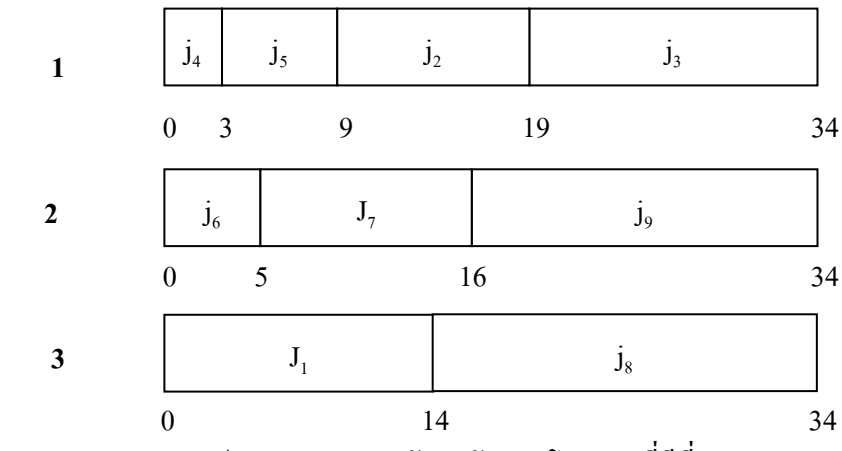
งาน	4	6	5	2	7	1	3	9	8
เวลา	3	5	6	10	11	14	15	18	20

หน่วยประมวลผลที่



รูป 4.4 แสดงการจัดลำดับงานโดยขั้นตอนวิธีละโมบ

หน่วยประมวลผลที่



รูป 4.5 แสดงการจัดลำดับงานโดยรวมที่ดีที่สุด

จากรูปที่ 4.4 จะเห็นว่าจะได้เวลาแล้วเสร็จโดยรวมเป็น 36 แต่เวลาแล้วเสร็จโดยรวมที่ดีที่สุดเป็น 34 ดังรูปที่ 4.5 ซึ่งจะต้องใช้อัลกอริทึมที่ซับซ้อนขึ้นและใช้เวลาในการจัดลำดับที่นานขึ้น ซึ่งจะเห็นว่า หากไม่จำเป็นต้องการการจัดลำดับงานที่ดีที่สุด ขั้นตอนวิธีละโมบก็เป็นทางเลือกที่ดีในการจัดลำดับงาน เพราะให้เวลาแล้วเสร็จโดยรวมที่ดีและเป็นวิธีการที่ง่าย

4.3 ปัญหาการเลือกกิจกรรม (activity selection problem)

กำหนดให้ มีเซ็ตของกิจกรรม n กิจกรรมที่ต้องการใช้ทรัพยากร ซึ่งอาจเป็นห้องโถง ห้องบรรยาย ห้องปฏิบัติการ สนามกีฬา และอื่น ๆ โดยที่ ณ ขณะใดขณะหนึ่งสามารถทำกิจกรรมได้เพียงกิจกรรมเดียว ซึ่งแต่ละกิจกรรมมีกำหนดเวลาที่เริ่มต้น (s_i) และเวลาแล้วเสร็จ (f_i) เราสามารถเลือกกิจกรรมให้แก่ทรัพยากรดังกล่าวได้ดังนี้

1. ทำการเรียงลำดับเวลาแล้วเสร็จของแต่ละกิจกรรมจากน้อยไปมาก
2. สำหรับแต่ละกิจกรรมที่เรียงลำดับตามข้อ 1
   2.1 นำกิจกรรมที่ 1 ใส่ลงใน SelectA = เซ็ตของกิจกรรมที่เลือกให้กับทรัพยากร
   2.2 นำกิจกรรมที่ 2, ..., n มาพิจารณา หากไม่ซ้อนทับกับเวลาของกิจกรรมอื่นใน A ให้นำกิจกรรมดังกล่าวเพิ่มลงใน SelectA

ตัวอย่าง 4.4 กำหนดให้ กิจกรรม 12 กิจกรรมที่ต้องการใช้ห้องประชุมห้องหนึ่ง โดยกิจกรรมต่าง ๆ มีแล้วเริ่มต้น (s_i) และเวลาแล้วเสร็จ (f_i) ดังตารางข้างล่างนี้

กิจกรรม i	1	2	3	4	5	6	7	8	9	10	11	12
s_i	0	2	3	1	8	8	6	5	3	6	12	0
f_i	6	13	5	4	11	12	7	9	8	10	14	2

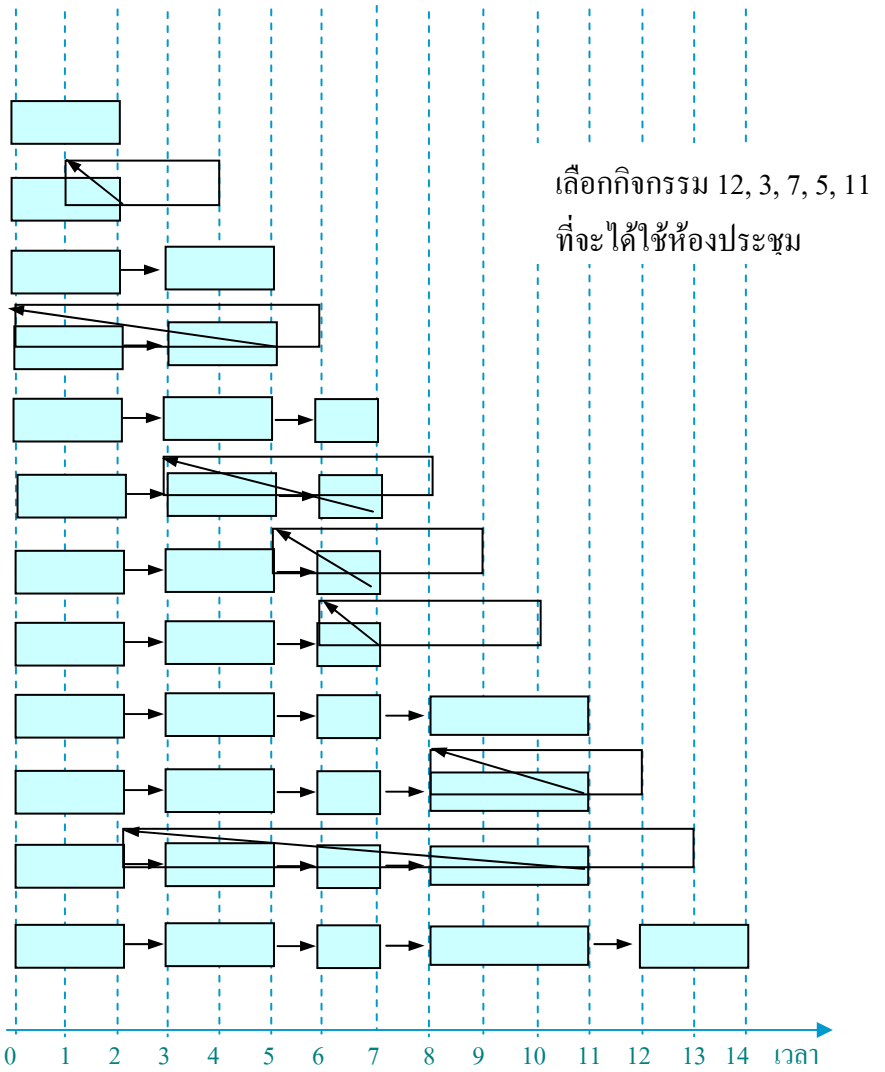
โปรแกรม4.1 ActivitySelect.java

```
public class ActivitySelect
{   static int act[] ={0,12,4,3,1,7,9,8,10,5,6,2,11};
    static int selectA[] = new int[51];
    static int start[] = {0,0, 1, 3, 0, 6, 3, 5, 6, 8, 8, 2, 12};
    static int finish[] = {0,2,4,5,6, 7, 8, 9, 10,11, 12, 13, 14 };
    static int n=12;
    static int k=1;
    public static void GreedyActivitySelector(int [] s, int [] f)
     {  int i,m;
        i=1;
        selectA[k]=act[1];
         for (m=2;m<=n;++m)
```

```
        {    if(s[m] >= f[i])
               { k=k+1;
                 selectA[k]=act[m];
                i=m;
                }//if
            }//next m
   } //Greedy
//-----------------------------------------------------------------
  public static void main(String[] args)
  {    int i;
       GreedyActivitySelector(start, finish);
       for(i=1;i<=k;++i)
         {  System.out.println(selectA[i]);  }//next i
  }//end main
}//end class
```

ทำการเรียงลำดับตามเวลาแล้วเสร็จได้ดังนี้

i	s_i	f_i
12	0	2
4	1	4
3	3	5
1	0	6
7	6	7
9	3	8
8	5	9
10	6	10
5	8	11
6	8	12
2	2	13
11	12	14



รูป 4.6 ปัญหาการเลือกกิจกรรม

4.4 ปัญหาการจัดลำดับงาน (Task scheduling problem)

เป็นการจัดลำดับงานสำหรับเครื่องจักรเดียวโดยแต่ละงานจะใช้เวลาในการผลิต 1 วัน เท่ากัน และแต่ละงานจะมีกำหนดเส้นตาย และค่าปรับหากส่งงานไม่ทันกำหนด เป้าหมายก็คือ จะจัดลำดับงานอย่างไรให้ได้จำนวนงานที่มากที่สุดและจ่ายค่าปรับน้อยที่สุด กำหนดให้

- เซ็ต S = $\{a_1, a_2, a_3,\ldots,a_n\}$ เป็นงานที่ 1, 2,…, n
- เซ็ต D = $\{d_1, d_2, d_3,\ldots,d_n\}$ เป็นเส้นตายของงานที่ 1, 2,…, n
- เซ็ต P = $\{p_1, p_2, p_3,\ldots..p_n\}$ เป็นค่าปรับหากส่งงานไม่ทันกำหนดของงานที่ 1, 2,…, n

ตัวอย่าง 4.5 จงจัดลำดับงานตามตาราง เพื่อให้งานเสร็จภายในกำหนดเส้นตาย และจ่ายค่าปรับน้อยที่สุด

<u>วิธีทำ</u> โดยขั้นตอนวิธีละโมบ ให้ทำดังนี้

1. เรียงลำดับค่าปรับจากมากไปน้อย

	Task						
a_1	1	2	3	4	5	6	7
d_1	4	2	4	3	1	4	6
p_1	70	60	50	40	30	20	10

2. เลือกงานที่สามารถทำเสร็จทันเวลา ในที่นี้จะเลือกทำ $\{a_1, a_2, a_3, a_4, a_7\}$ ก่อน แล้วค่อยทำงาน $\{a_5, a_6\}$ ทีหลัง
3. จัดเรียงลำดับงานใหม่ตามเส้นตาย จะได้ $\{ a_2, a_4, a_1, a_3, a_7, a_5, a_6\}$
4. จ่ายค่าปรับเท่ากับ $p_5, p_6 = 30+20 = 50$

   <u>complexity</u> = $O(n^2)$

4.5 ปัญหาถุงเป้(knapsack problem)

ปัญหาถุงเป้สนใจว่า ถ้ามีขโมยเอาถุงเป้เข้าไปในร้านค้าเพื่อขโมยของ เขาจะเลือกขโมยอะไรดีที่จะทำให้ได้มูลค่าสูงสุด โดยที่ถุงเป้ของเขาสามารถบรรจุของได้ไม่เกินน้ำหนัก W ซึ่งแบ่งเป็น 2 ปัญหา คือ

1. แบบเลือกและไม่เลือกใส่เป้ (0-1 kanapsack) ของที่ขโมยไม่สามารถแบ่งส่วนได้ ต้องเลือกที่หยิบทั้งชิ้นหรือไม่หยิบชิ้นนั้นเลย
2. เลือกหยิบบางส่วนใส่เป้ (fractional knapsack) สามารถที่จะเลือกหยิบบางส่วนของรายการนั้นได้

ตัวอย่าง 4.6 มีของ 3 ชิ้น โดยชิ้นที่1 หนัก 5 กก. มูลค่า 5,000 บาท ชิ้นที่2 หนัก 10 กก. มูลค่า 6,000 บาท และ ชิ้นที่ 3 หนัก 20 กก.มูลค่า 14,000 บาท ถุงเป้บรรจุน้ำหนักได้ไม่เกิน 30 กก. โดยขั้นตอนวิธีละโมบ เราจะมาคำนวณมูลค่าต่อกก. ของแต่ละรายการ

ชิ้นที่ 1 = 5,000/5 = 1,000 บาท/กก.

ชิ้นที่ 2 = 6,000/10 = 600 บาท/กก.

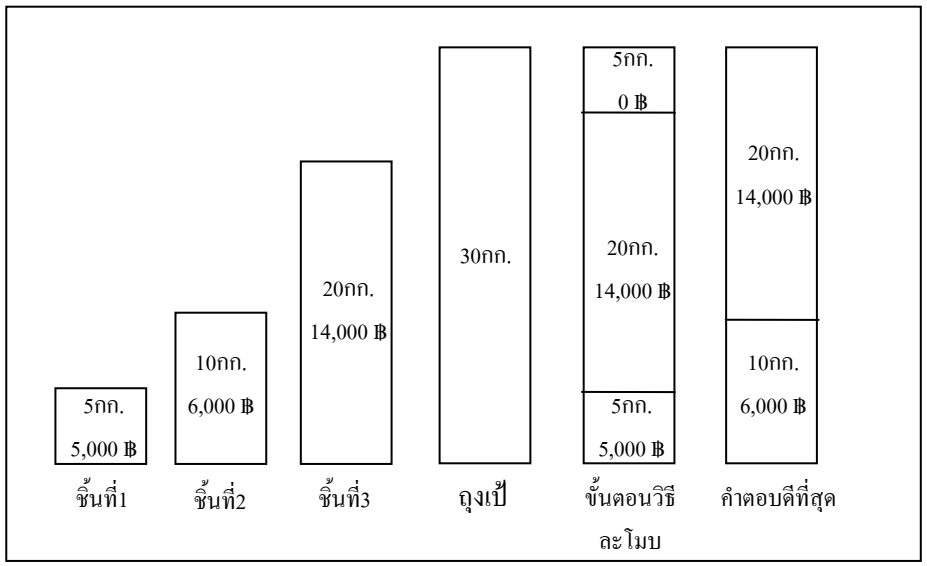
ชิ้นที่ 3 = 14,000/20 = 700 บาท/กก.

แล้วเรียงลำดับมูลค่าจากมากไปน้อย คือ ชิ้น1, ชิ้น3, ชิ้น2 แล้วหยิบของตามมูลค่ามากก่อน จนกว่าจะใส่ถุงไม่ได้ นั่นคือเราจะเลือกหยิบ ชิ้น1 กับ ชิ้น3

รวมมูลค่า= 5,000+14,000 = 19,000 บาท น้ำหนัก = 5+20 =25 กก. (เหลือที่ว่าง 5 กก.)

ทั้งที่ถ้าเลือกหยิบชิ้น 2 กับชิ้น3 จะได้มูลค่ามากกว่า

รวมมูลค่า= 6,000+14,000 = 20,000 บาท น้ำหนัก = 10+20 =30 กก.



รูป 4.7 เปรียบเทียบขั้นตอนวิธีละโมบกับกำหนดการพลวัต สำหรับปัญหาถุงเป้ 0-1

คราวนี้ลองใช้ขั้นตอนวิธีละโมบกับปัญหาเลือกหยิบบางส่วนใส่เป้ เราจะหยิบ

ชิ้นที่1 หนัก 5 กก. มูลค่า 5,000บาท +

ชิ้นที่3 หนัก 20 กก. มูลค่า 14,000 บาท +

ชิ้นที่2 อีก 5 กก. มูลค่า 3,000 บาท รวมเป็น 22,000 บาท

สรุปว่าขั้นตอนวิธีละโมบใช้ได้ดีกับปัญหาถุงเป้แบบเลือกหยิบบางส่วนใส่เป้ แต่จะใช้ไม่ได้กับปัญหาถุงเป้แบบเลือกและไม่เลือกใส่เป้ ซึ่งต้องใช้วิธีกำหนดการพลวัต โดย

$$P[i][w] = \begin{cases} 0 & if\ w = 0 \\ \max(P[i-1][w], p_i + P[i-1][w-w_i]) & if\ w_i \le w \\ p[i-1][w] & if\ w_i > w \end{cases}$$

Complexity : ขั้นตอนวิธีละโมบ ขึ้นอยู่กับอัลกอริทึมที่ใช้เรียงลำดับเช่น merge sort = O(nlogn)

Complexity : กำหนดการพลวัต กับ 0-1 knapsack problem = $O(2^n)$

4.6 การบีบอัดข้อมูลโดยใช้ Huffman Code (data compression by Huffman Code)

ในหัวข้อนี้ เราจะพิจารณาการประยุกต์ใช้ขั้นตอนวิธีละโมบในการบีบอัดข้อมูลของไฟล์ โดยทั่วไป รหัสแอสกีที่ใช้แทนตัวอักขระต่าง ๆ จะใช้ 8 บิต

สมมติ เรามีไฟล์ซึ่งประกอบด้วยตัวอักขระและความถี่ดังตาราง

ตาราง 4.1 ตัวอักขระและความถี่ที่บรรจุในไฟล์ที่ต้องการบีบอัดข้อมูล

ตัวอักขระ	A	E	I	S	T	Space	Comma
ความถี่	10	15	12	3	4	13	1

เนื่องจากเรามี 7 อักขระจึงต้องใช้ 3 บิต ($2^3 = 8$)ในการแทนแต่ละอักขระ ซึ่งหากเรากำหนดให้ทุกอักขระใช้ 3 บิตเท่ากัน เราจะกำหนดรหัสสำหรับแต่ละอักขระและคำนวณจำนวนบิตทั้งหมดที่ต้องใช้ดังตาราง 4.2

ตาราง 4.2 การให้รหัสแทนแต่ละอักขระโดยใช้จำนวนบิตเท่ากัน

ตัวอักขระ	รหัส	ความถี่	จำนวนบิตทั้งหมด
A	000	10	30
E	001	15	45
I	010	12	36
S	011	3	9
T	100	4	12
Space	101	13	39
Comma	110	1	3
รวม			174

จากตารางที่ 4.2 เราต้องใช้จำนวนบิตทั้งหมด 174 บิตในการแทนอักขระทั้งหมดในไฟล์ เนื่องจากความถี่ของแต่ละอักขระแตกต่างกัน จึงเกิดแนวคิดที่ว่าหากเรากำหนดให้แต่ละอักขระแทนด้วยจำนวนบิตที่ไม่เท่ากัน โดยอักขระที่มีความถี่สูงจะแทนด้วยจำนวนบิตที่น้อย และอักขระที่มีความถี่ต่ำจะแทนด้วยจำนวนบิตที่มากขึ้น อาจทำให้สามารถประหยัดจำนวนบิตทั้งหมดที่แทนอักขระทั้งหมดในไฟล์ได้มากขึ้น

แนวคิดของการใช้จำนวนบิตที่ไม่เท่ากันในการแทนตัวอักขระคิดขึ้นโดย Huffman ใช้ขั้นตอนวิธีละโมบ โดยวิธีการของ Huffman ใช้ต้นไม้ทวิภาคที่มีกิ่งทางซ้ายแทนด้วยรหัส 0 และกิ่งทางขวาแทนด้วยรหัส 1 โดยมี Q แทนคิวที่มีลำดับความสำคัญในการเก็บตัวอักขระและ

ความถี่ของอักขระดังกล่าว โดยกำหนดให้ ความถี่ต่ำมีลำดับความสำคัญมาก วิธีการของ Huffman พอสรุปได้ดังนี้

1. ดึงตัวอักขระ 2 ตัวที่มีความถี่น้อยที่สุดออกจาก Q และนำมาสร้างเป็นต้นไม้ทวิภาค โดยให้ตัวอักขระแรกที่ดึงจาก Q เป็นกิ่งทางซ้าย และตัวอักขระตัวที่สองเป็นกิ่งทางขวา
2. ทำการรวมความถี่ของกิ่งทางซ้ายและกิ่งทางขวาเข้าด้วยกันแล้วนำไปเพิ่มเข้าไปใน Q
3. ทำซ้ำข้อ 1, 2 จนกระทั่ง Q เป็นคิวว่าง

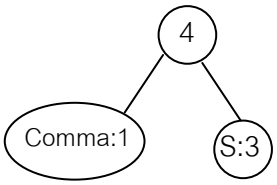
ตัวอย่าง 4.7 จงใช้ Huffman Code ในการบีบอัดข้อมูลในไฟล์ที่มีข้อมูลดังตาราง

วิธีทำ

เรียงลำดับข้อมูลจากน้อยไปมากโดยใช้คิวที่มีลำดับความสำคัญ

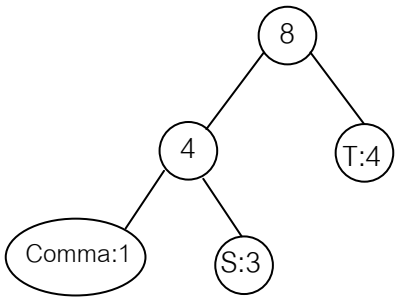
ตัวอักขระ	Comma	S	T	A	I	Space	E
ความถี่	1	3	4	10	12	13	15

ดึงข้อมูลที่มีความถี่น้อยที่สุดและน้อยรองลงมา ในที่นี้คือ Comma :1 และ S:3 โดยให้ข้อมูลที่น้อยที่สุด Comma :1 เป็นลูกทางซ้ายและน้อยรองลงมา S:3 เป็นลูกทางขวา และนำความถี่ของข้อมูลทางซ้ายและทางขวามารวมกันดังนี้



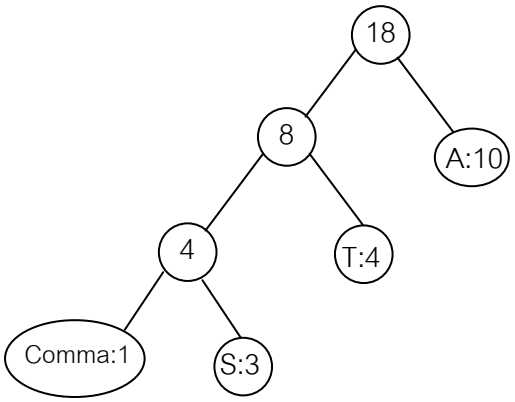
ตัวอักขระ	Comma, S	T	A	I	Space	E
ความถี่	4	4	10	12	13	15

ดึงข้อมูลที่มีความถี่น้อยที่สุดและน้อยรองลงมา ในที่นี้คือ Comma, S :4 และ T:4 โดยให้ข้อมูลที่น้อยที่สุด Comma, S :4 เป็นลูกทางซ้ายและน้อยรองลงมา T:4 เป็นลูกทางขวา และนำความถี่ของข้อมูลทางซ้ายและทางขวามารวมกันดังนี้



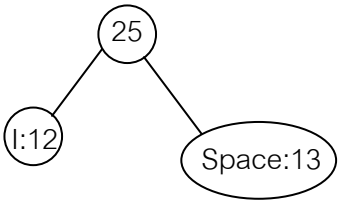
ตัวอักขระ	Comma, S,T	A	I	Space	E
ความถี่	8	10	12	13	15

ดึงข้อมูลที่มีความถี่น้อยที่สุดและน้อยรองลงมา ในที่นี้คือ Comma, S, T : 8 และ A:10 โดยให้ข้อมูลที่น้อยที่สุด Comma, S, T : 8 เป็นลูกทางซ้ายและน้อยรองลงมา A:10 เป็นลูกทางขวา และนำความถี่ของข้อมูลทางซ้ายและทางขวามารวมกันดังนี้



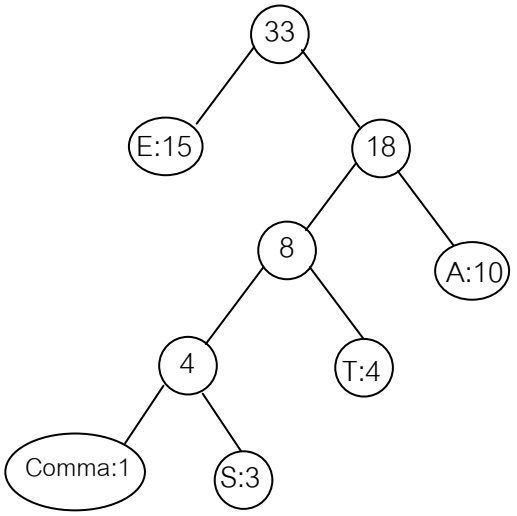
ตัวอักขระ	I	Space	E	Comma, S,T, A
ความถี่	12	13	15	18

ดึงข้อมูลที่มีความถี่น้อยที่สุดและน้อยรองลงมา ในที่นี้คือ I : 12 และ Space:13 โดยให้ข้อมูลที่น้อยที่สุด I : 12 เป็นลูกทางซ้ายและน้อยรองลงมา Space:13 เป็นลูกทางขวา และนำความถี่ของข้อมูลทางซ้ายและทางขวามารวมกันดังนี้



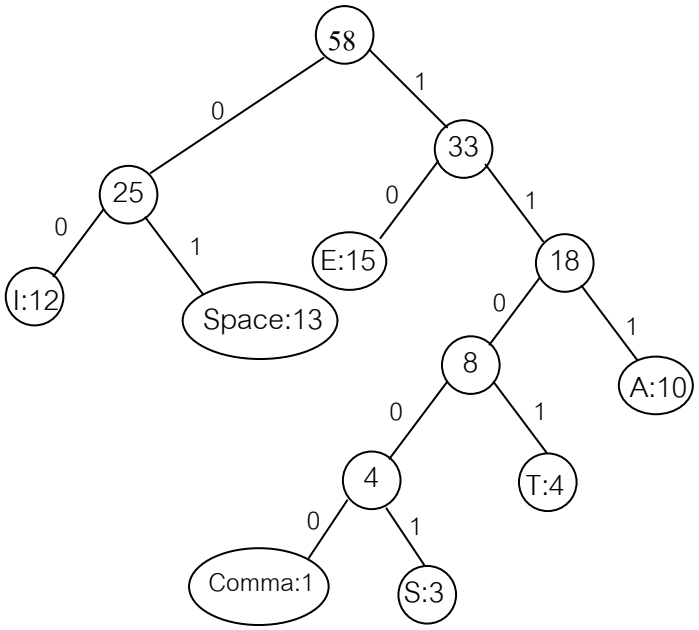
ตัวอักขระ	E	Comma, S,T, A	I, Space
ความถี่	15	18	25

ดึงข้อมูลที่มีความถี่น้อยที่สุดและน้อยรองลงมา ในที่นี้คือ E : 15 และ Comma, S,T, A:13 โดยให้ข้อมูลที่น้อยที่สุด E : 15 เป็นลูกทางซ้ายและน้อยรองลงมา Comma, S,T, A:13 เป็นลูกทางขวา และนำความถี่ของข้อมูลทางซ้ายและทางขวามารวมกันดังนี้



ตัวอักขระ	I, Spaxe	E,Comma, S,T, A
ความถี่	25	33

ดึงข้อมูลที่มีความถี่น้อยที่สุดและน้อยรองลงมา ในที่นี้คือ I, Space : 25 และ E, Comma, S,T, A:33 โดยให้ข้อมูลที่น้อยที่สุด I, Space : 25 เป็นลูกทางซ้ายและน้อยรองลงมา E, Comma, S,T, A:33 เป็นลูกทางขวา และนำความถี่ของข้อมูลทางซ้ายและทางขวามารวมกันดังนี้



<u>รูป 4.8</u> Huffman's tree

ตาราง 4.3 การให้รหัสแทนแต่ละอักขระโดยใช้วิธี Huffman Code

ตัวอักขระ	รหัส	ความถี่	จำนวนบิตทั้งหมด
A	111	10	30
E	10	15	30
I	00	12	24
S	11001	3	15
T	1101	4	16
Space	01	13	26
Comma	11000	1	5
รวม			146

จากโจทย์ข้างต้นจะเห็นว่าวิธี Huffman Code ใช้จำนวนบิตทั้งหมดในการแทนข้อมูลน้อยกว่าวิธีการให้รหัสแทนแต่ละอักขระโดยใช้จำนวนบิตเท่ากัน นั่นก็คือ การบีบอัดข้อมูลโดยวิธี Huffman Code ทำให้ได้ไฟล์ขนาดเล็กลง ซึ่งหากไฟล์ใดประกอบด้วยตัวอักขระจำนวนไม่มากนัก และมีความถี่สูงมากๆ จะสามารถบีบอัดไฟล์ได้ขนาดเล็กกว่าเดิมมาก ในทำนองเดียวกัน หากไฟล์ใดประกอบด้วยตัวอักขระจำนวนมาก และมีความถี่น้อย การบีบอัดข้อมูลจะทำให้ได้ไฟล์ซึ่งมีขนาดไม่แตกต่างจากเดิมมากนัก

Complexity = O(n log n)

โปรแกรม 4.2 Huffman.java

```
/*
* Note: this program only process text characters:
 *       ('a'-'z' / 'A'-'Z'). Everything else is ignored.
 */
import java.io.*;
import java.util.*;
class Node
   implements Comparable
{
       private int     value;
       private char    content;
       private Node    left;
       private Node    right;
       public Node(char content, int value) //constructor
```

```
    {   this.content  = content;
        this.value    = value;   }
    public Node(Node left, Node right)
    {
        // Assumes that the left three is always the one that is lowest
        this.content  = (left.content < right.content) ? left.content : right.content;
        this.value    = left.value + right.value;
        this.left      = left;
        this.right    = right;
    }
    public int compareTo(Object arg)
    {
        Node other = (Node) arg;
        // Content value has priority and then the lowest letter
        if (this.value == other.value)
            return this.content-other.content;
        else
            return this.value-other.value;
    }
//---------------------------------------------------------------------------
    private void printNode(String path)
    {
        if ((left==null) && (right==null))
            System.out.println(content + " " + path);
        if (left != null)
            left.printNode(path + '0');
        if (right != null)
            right.printNode(path + '1');
    }
//---------------------------------------------------------------------------
    public static void printTree(Node tree)
    {
        tree.printNode("");
    }
}// end class Node
//-------------------------------------------------------------
public class Huffman
{
```

```
public static void main(String[] args) throws IOException
{    StringBuffer fileContents = new StringBuffer();
     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
     String line = null;
     while ((line = br.readLine()) != null)
          fileContents.append("\n").append(line);
     processFile(fileContents.toString());
}
private static void processFile(String fileContents)
{
     int[] frequency = new int['Z'-'A'+1];          // Frequency table of each letter
     TreeSet<Node> trees new TreeSet<Node>();  //ist containing all trees -- ORDERED!
     // Build the frequency table of each letter
     for (int i=0; i<fileContents.length(); i++)
     {
          char ch = Character.toUpperCase(fileContents.charAt(i));
          if ((ch >= 'A') && (ch <= 'Z'))
               ++frequency[ch - 'A'];
     }
     // Build up the initial trees
     for (int i=0; i<'Z'-'A'+1; i++)
     {
          if (frequency[i] > 0)
          {
               Node n = new Node((char)('A'+i), frequency[i]);
               trees.add(n);
          }
     }
     // Huffman algoritm
     while (trees.size() > 1)
     {
          Node tree1 = (Node) trees.first();
          trees.remove(tree1);
          Node tree2 = (Node) trees.first();
          trees.remove(tree2);
          Node merged = new Node(tree1, tree2);
          trees.add(merged);
     }
```

```
            // Print the resulting tree
            if (trees.size() > 0)
            {
                Node theTree = (Node) trees.first();
                Node.printTree(theTree);
            }
            else
                System.out.println("The file didn't contain useful characters.");
        }
}
```

Outputs

```
sssssssssssssssssdjjjjjjjjjjjjjjjjkaaaaaaaaaaaaaaaajsiiiiiiiiiiwqooooooooodkkkk
kkkkjjjjjjjjjjjjwwwwwwwwwwwwwwwwwiiiiiiiiiiiiiiireeeeeeeeeeeeeeeeeeeeeee
J 00
A 010
W 011
S 100
O 1010
D 101100
Q 1011010
R 1011011
K 10111
E 110
I 111
```

แบบฝึกหัด

1. แฟ้มข้อมูลประกอบด้วยตัวอักษร A, B, I, M, S, X, Z ด้วยความถี่ตามลำดับดังนี้ A(12), B(7), I(18), M(10), S(9), X(5), Z(2) จงหา Huffman's tree & Huffman code

2. จาก Huffman code ในข้อ 1 ให้แปลง bit string ต่อไปนี้เป็น ตัวอักษร

   (ก) 01100010101010

   (ข) 1000100001010

   (ค) 11100100111101

   (ง) 1000010011100

3. ใช้ขั้นตอนวิธีของ Huffman เพื่อสร้าง Huffman code สำหรับข้อมูลต่อไปนี้

letter	c	e	i	r	s	t	x
probability	.11	.22	.16	.12	.15	.10	.14

4. จากงาน และเวลาที่ใช้ในแต่ละงาน จงจัดลำดับงานให้ได้ average complete of time น้อยที่สุด

งานที่	เวลาที่ใช้
1	7
2	3
3	10
4	5

5. พิจารณาลำดับงานที่มีกำหนดเส้นตาย และรายได้ต่อไปนี้ จากนั้นจัดลำดับงานเพื่อให้มีรายได้และเสร็จทันเวลามากที่สุด

Job	deadline	รายได้
1	2	40
2	4	15
3	3	60
4	2	20
5	3	10
6	1	45
7	1	55

6. ถ้ามีของ n ชิ้น แต่ละชิ้นมีความยาว s_i และ น้ำหนัก w_i ต้องการหยิบของใส่เป้ที่มีความยาวเท่ากับ S ให้เต็มโดยสามารถเลือกหยิบบางส่วนได้ และต้องการให้น้ำหนักรวมของถุงเป้เบาที่สุด จงแก้ปัญหาสำหรับข้อมูลต่อไปนี้

(ก) S=20, s_1=9, s_2=5, s_3=7, s_4=12, s_5=3 และ w_1=4, w_2=4, w_3=8, w_4=5, w_5=1

(ก) S=18, s_1=9, s_2=5, s_3=7, s_4=12, s_5=3 และ w_1=18, w_2=15, w_3=28, w_4=50, w_5=6

(ก) S=25, s_1=12, s_2=6, s_3=17, s_4=32, s_5=23 และ w_1=18, w_2=15, w_3=28, w_4=50, w_5=6