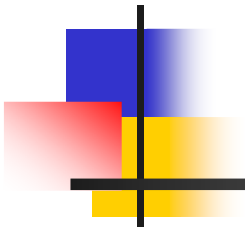


Chapter 10

Binary Tree Application : Huffman Coding

Huffman Coding

- **Huffman Coding**
  - ใช้สำหรับบีบอัดข้อมูลก่อนส่งไปยังปลายทาง เพื่อลดขนาดของข้อมูลที่ส่ง
  - **Proposed by Dr. David A. Huffman in 1952,** *"A Method for the Construction of Minimum Redundancy Codes"*
  - **Statistical-Base** ต้องวิเคราะห์ความถี่ของการปรากฏของตัวอักษร ทั้งหมดก่อนแล้ว สร้างรหัสใหม่สำหรับตัวอักษรแต่ละตัว
  - เข้ารหัสแบบ **variable length** รหัสแต่ละตัวอักษรจะยาวไม่เท่ากัน
  - ตัวอักษรที่ใช้บ่อยครั้งมากกว่า จะมีคำรหัสที่สั้นกว่า
- **Huffman Algorithm**
  - นับจำนวนความถี่ข้อมูล(Text) ที่จะถูกบีบอัดทุกตัว
  - เรียงลำดับความสำคัญตามความถี่ที่เกิดขึ้นของตัวอักษร (Priority Queue)
  - สร้าง **Huffman code tree** ตามลำดับความสำคัญ(ความถี่)ของตัวอักษร
    - จับคู่ **node** ที่มีความถี่น้อยที่สุดเข้าด้วยกันทีละคู่ (ดึงคิว 2 ตัวแรก)แล้วใส่กลับเข้าไปใน **Queue** ทำจนเหลือตัวเดียว
  - สำรวจเส้นทาง (Traversal) ของแต่ละตัวอักษร นำมาสร้างเป็นตารางรหัส
  - เปลี่ยนตัวอักษรใน **Text** ทั้งหมดด้วยบิทของรหัสที่ได้

"คนขายของจะขายของหลากหลาย"

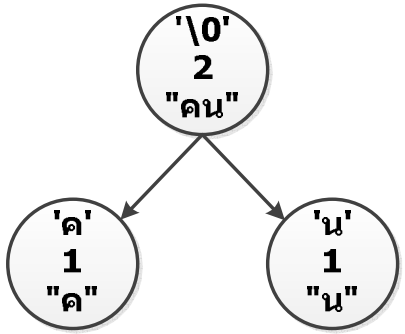
char	count
ค	1
น	1
ข	4
า	4
ย	3
อ	2
ง	2
จ	1
ห	2
ล	2
ะ	1
ก	1

สร้างตาราง(ตัวอักษร, ความถี่ที่เกิด)

```
LinkedHashMap<Character, Integer> table = new LinkedHashMap<Character, Integer>();
```

```
static void countText(LinkedHashMap<Character, Integer> table, String text) {
      for (int i = 0; i < text.length(); i++) {
            char a = text.charAt(i); //ดึงตัวอักษรทีละตัว
            if (table.containsKey(a)) //เช็คว่ามีในตาราง
                table.put(a, table.get(a) + 1); //ถ้ามีนับเพิ่ม+1
            else
                table.put(a, 1); //ถ้าไม่มีใส่เพิ่มแล้วนับ 1
           }
}
```

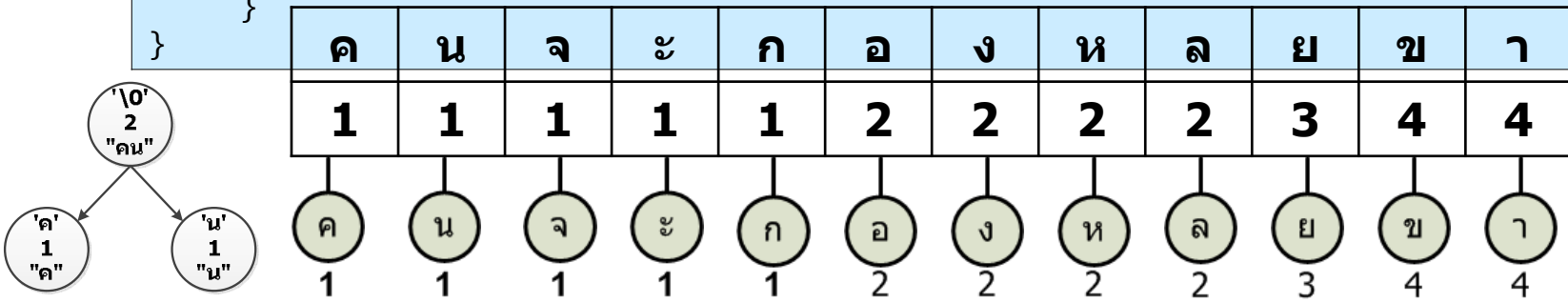
```
HuffmanNode {
  char alpha; //ตัวอักษร
  int freq; //ความถี่
  String str; //จับคู่โหนดตัวอักษร เพื่อสร้าง Tree
  HuffmanNode  left,  right; }
```



สร้างตารางเรียงลำดับความถี่

```
q ที่สร้างเพื่อจับคู่โหนดของตัวอักษร
PriorityQueue<HuffmanNode> q = new PriorityQueue<HuffmanNode>();
```

```
แปลงตารางความถี่เพื่อจับคู่ (ใช้ Insertion sort หรือ Priority Queue)
static void  convertMapToPriorityQ ( LinkedHashMap<Character, Integer> table ,PriorityQueue <HuffmanNode> q)
{    for(Character c : table.keySet()) {  //วนรอบทุกตัวอักษรในตารางความถี่
          q.add(new HuffmanNode (c, table.get(c)));  //นำข้อมูลจากตาราง สร้างโหนด และ add ใส่ คิว
     }
}
```

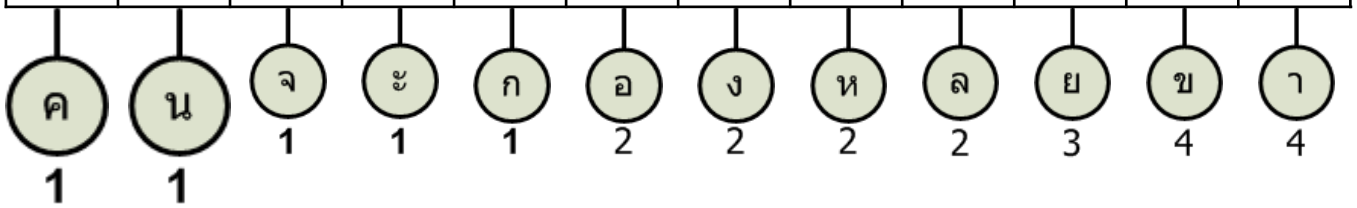



```
สร้าง Huffman Tree
HuffmanNode root = huffmanTree(q);
```

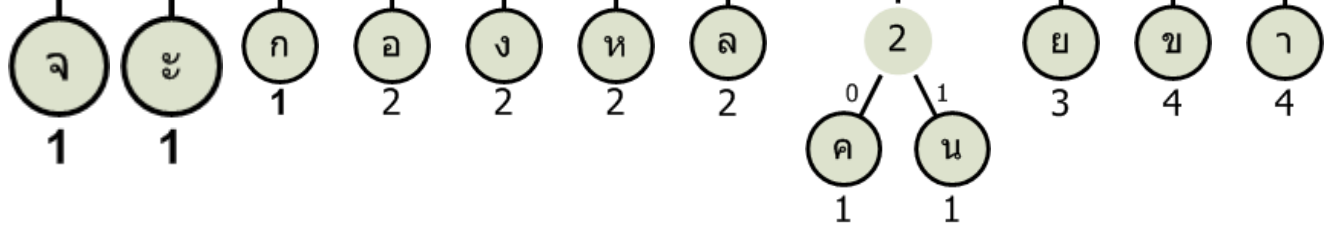
```
static HuffmanNode huffmanTree(PriorityQueue<HuffmanNode> q) {
    while (q.size() > 1) { // รวมจนเหลือต้นไม้ต้นเดียว
       HuffmanNode z = new HuffmanNode(); // สร้างโหนดใหม่ เพื่อรวม subtree ซ้ายและขวา
       z.left = q.poll();              // ดึง subtree จากคิว สร้างเป็นโหนดซ้าย
       z.right = q.poll();    // ดึง subtree จากคิว สร้างเป็นโหนดขวา
       z.freq = z.left.freq + z.right.freq; // คำนวณความถี่รวม
       z.str = z.left.str + z.right.str; //สร้างชื่อโหนดที่รวมกัน
       q.add(z);  // ใส่ subtree ลงในคิว
    }
    return q.poll(); // return tree ที่ทำเสร็จแล้ว
}
```

จับคู่โหนดที่มีค่าความถี่น้อยสุด

ค	น	จ	ะ	ก	อ	ง	ห	ล	ย	ข	า
1	1	1	1	1	2	2	2	2	3	4	4

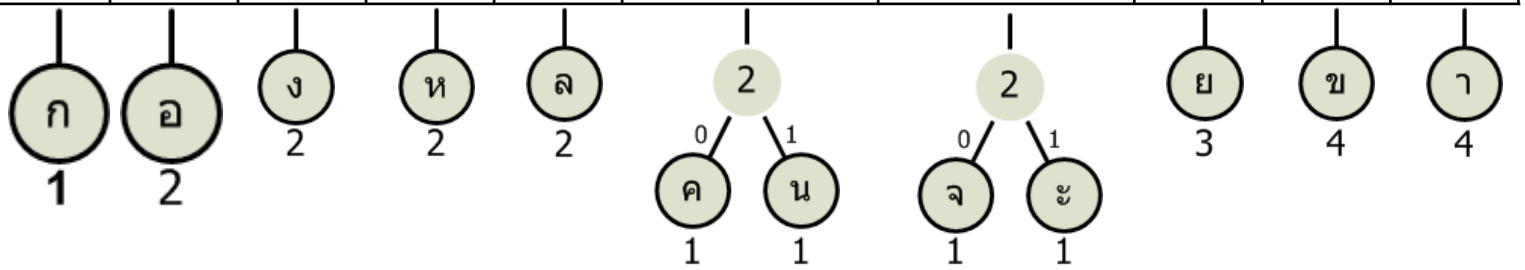


จ	ะ	ก	อ	ง	ห	ล	คน	ย	ข	า
1	1	1	2	2	2	2	2	3	4	4

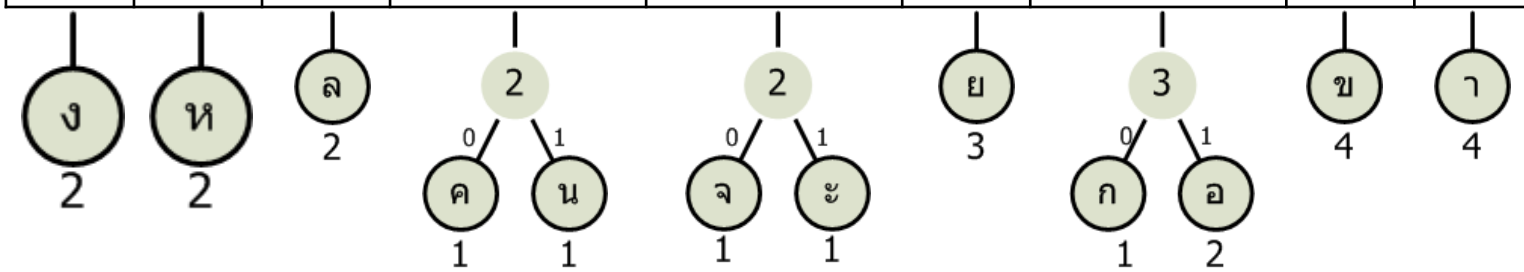


"จับคู่โหนดที่มีค่าความถี่น้อยสุด

ก	อ	ง	ห	ล	คน	จะ	ย	ข	า
1	2	2	2	2	2	2	3	4	4

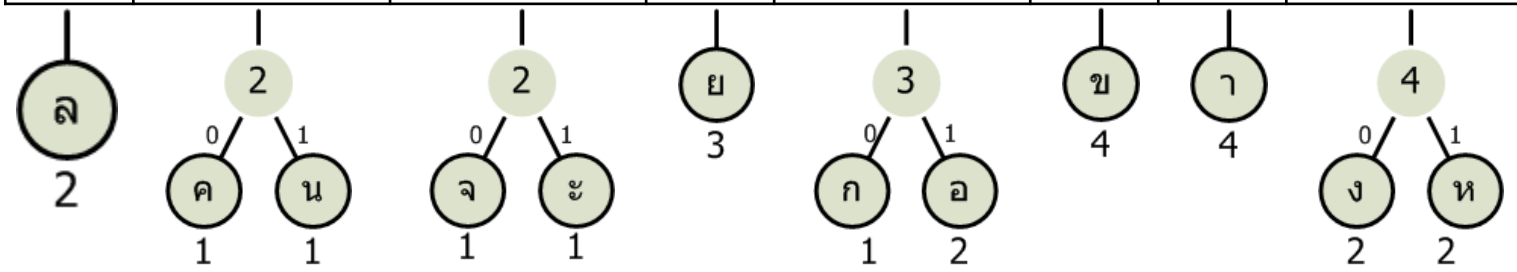


ง	ห	ล	คน	จะ	ย	กอ	ข	า
2	2	2	2	2	3	3	4	4

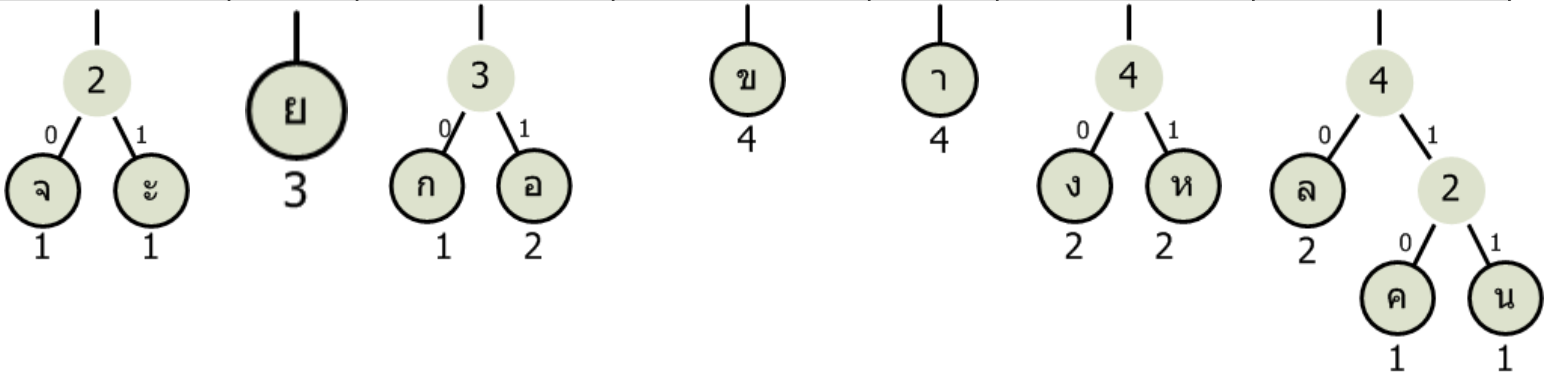


จับคู่โหนดที่มีค่าความถี่น้อยสุด

ล	คน	จะ	ย	กอ	ข	า	งห
2	2	2	3	3	4	4	4

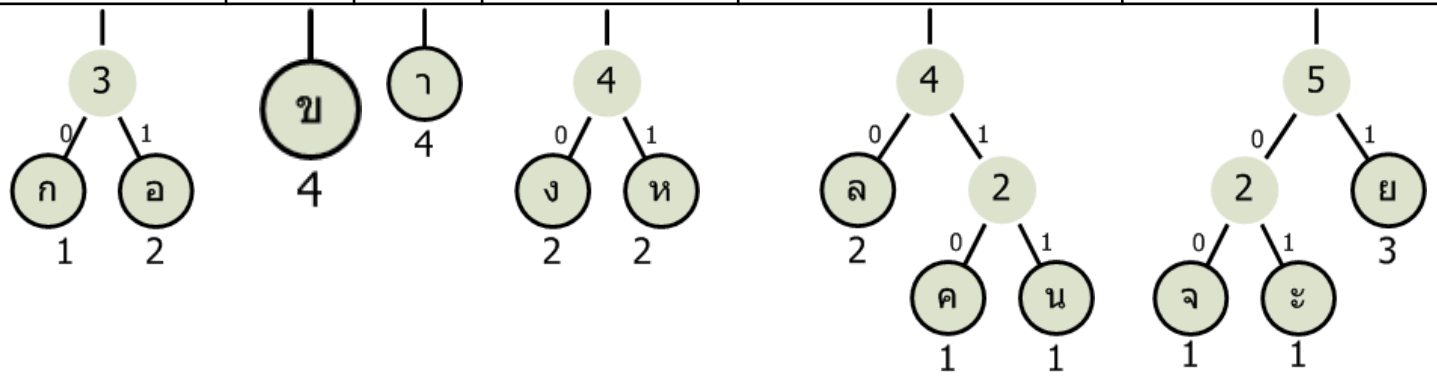


จะ	ย	กอ	ข	า	งห	ลคน
2	3	3	4	4	4	4

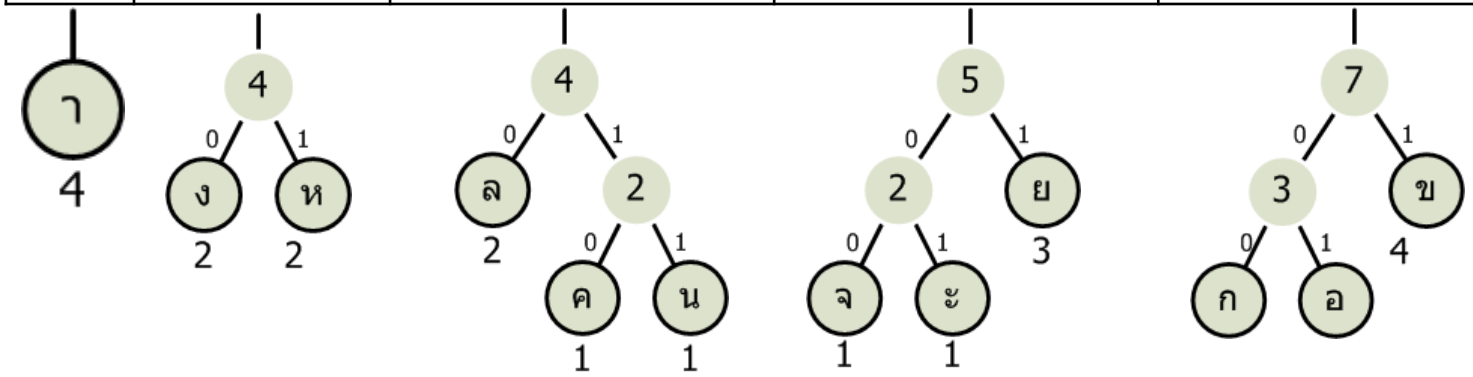


"จับคู่โหนดที่มีค่าความถี่น้อยสุด

กอ	ข	า	งห	ลคน	จะย
3	4	4	4	4	5

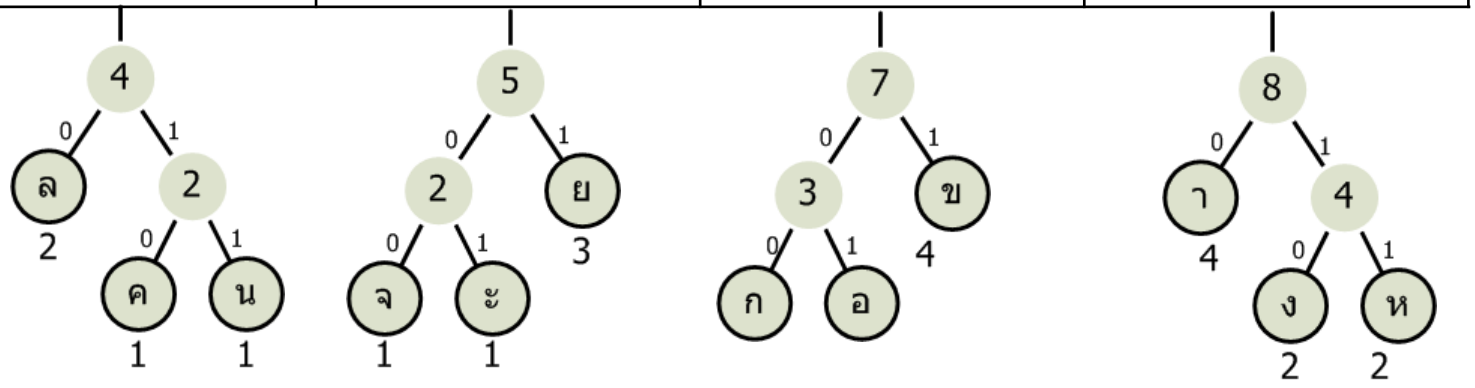


า	งห	ลคน	จะย	กอข
4	4	4	5	7

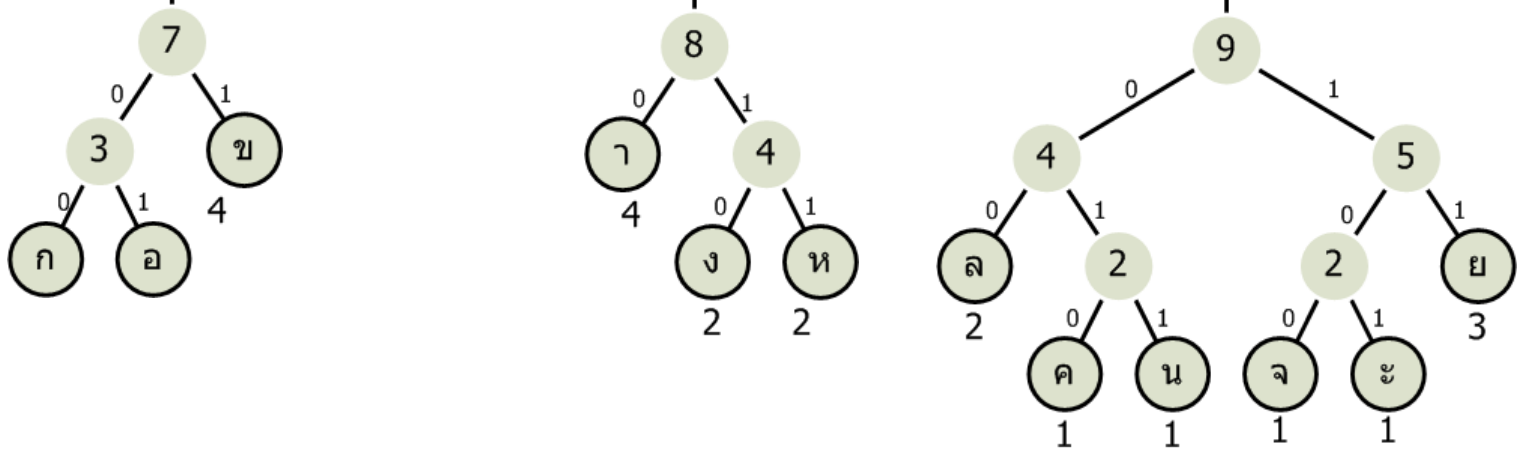


จับคู่โหนดที่มีค่าความถี่น้อยสุด

ลคน	จะย	กอข	างห
4	5	7	8

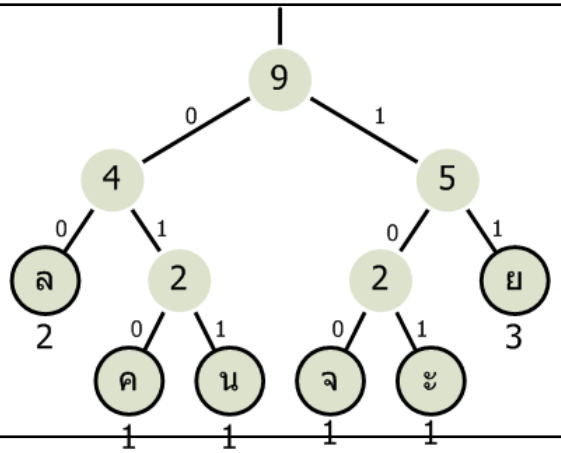


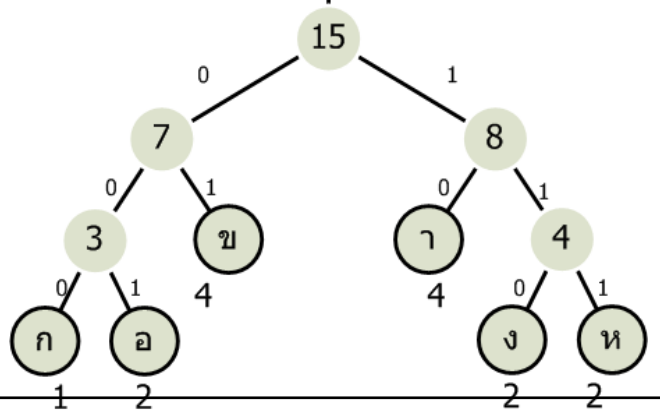
กอข	างห	ลคนจะย
7	8	9



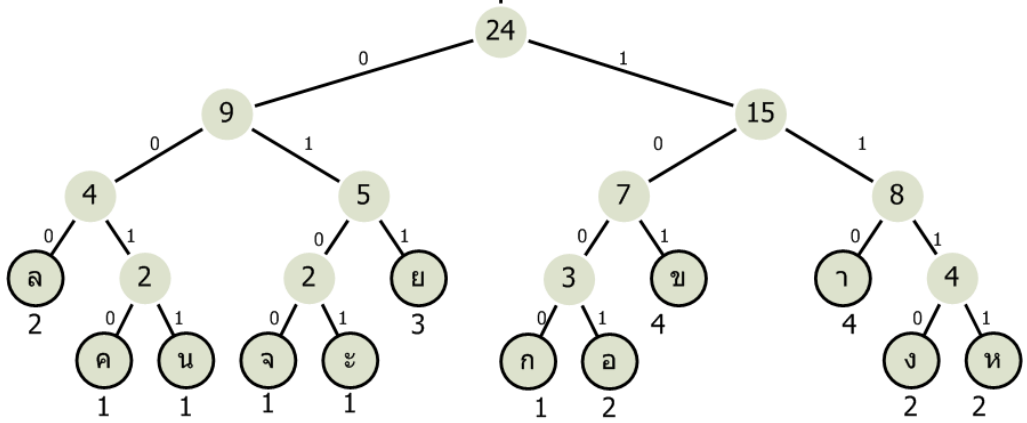
จนเหลือแค่ Tree เดียว

ลคนจะย	กอขางห
9	15

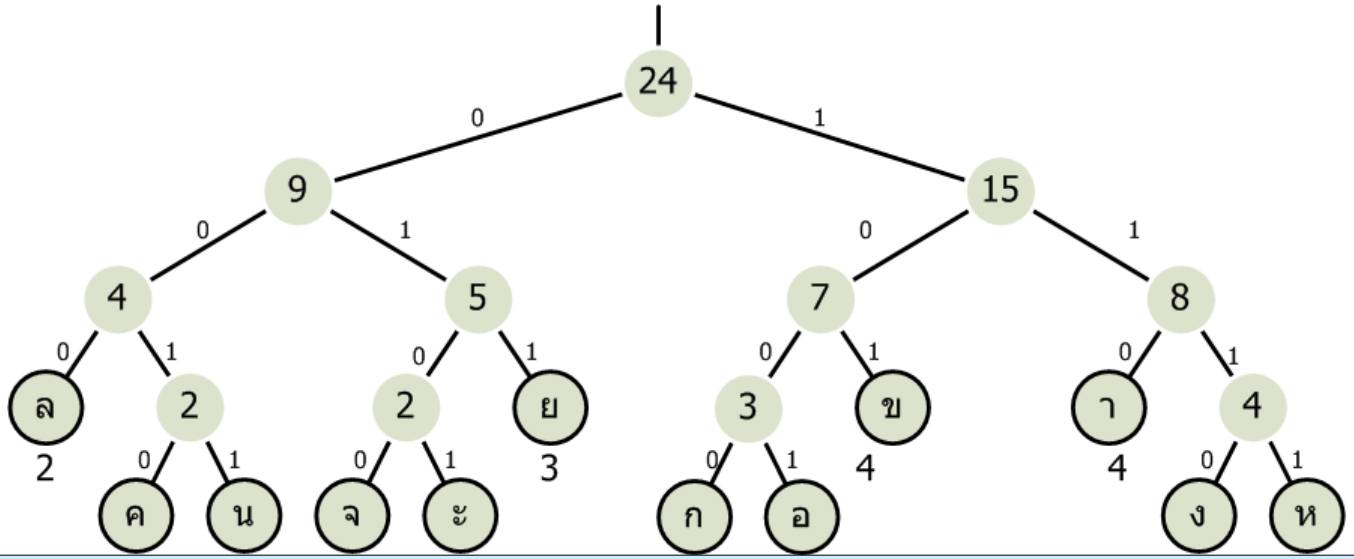




ลคนจะยกอขางห
24



สร้างตารางรหัส Huffman




```
สร้างตารางสำหรับถอดรหัส
static void createHuffmanTable(HuffmanNode root) {
    charToCode = new LinkedHashMap<Character, String>(); //เตรียมตาราง charToCode สร้างรหัส
    codeToChar = new LinkedHashMap<String, Character>(); //เตรียมตาราง codeToChar ถอดรหัส
    FillPostorderPath (root, new String());   //เติมรหัสลงในตาราง โดยใช้ recursion
}
static void FillPostorderPath (HuffmanNode n, String s) {//s คือ รหัสของ path ที่ได้
            if (n == null)  // Base case ไม่มีข้อมูล
                return;
            FillPostorderPath(n.left, s+"0"); //recursive case  เติมพาททางซ้ายด้วย 0
            FillPostorderPath(n.right, s+"1"); //recursive case  เติมพาททางขวาด้วย 1
            if (n.alpha != '\0') {  //กลับมาที่ตัวโหนดตัวอักษร
                charToCode.put(n.alpha, s);  //นำตัวอักษร และรหัสใส่ในตาราง charToCode
                codeToChar.put(s, n.alpha);  //นำรหัสและตัวอักษรใส่ในตาราง codeToChar
            }
}
```

Unique code table

	char	size of code	new code	Decimal code	count	total bits use
0	ล	3	00000000	0	2	6
1	ค	4	00000010	2	1	4
2	น	4	00000011	3	1	4
3	จ	4	00000100	4	1	4
4	ะ	4	00000101	5	1	4
5	ย	3	00000011	3	3	9
6	ก	4	00001000	8	1	4
7	อ	4	00001001	9	2	8
8	ข	3	00000101	5	4	12
9	า	3	00000110	6	4	12
10	ง	4	00001110	14	2	8
11	ห	4	00001111	15	2	8
					24	83

Unique code table

ตารางสำหรับเข้ารหัส
LinkedHashMap<Character, String> charToCode

char	String
ก	1000
ข	101
ค	0010
ง	1110
จ	0100
น	0011
ย	011
ล	000
ห	1111
อ	1001
ะ	0101
า	110

ตารางสำหรับถอดรหัส
LinkedHashMap<String, Character> codeToChar

String	char
1000	ก
101	ข
0010	ค
1110	ง
0100	จ
0011	น
011	ย
000	ล
1111	ห
1001	อ
0101	ะ
110	า

Huffman coding(binary bits)

- **แปลงข้อมูลเป็น Binary Sequence**

  ค | น | ข | า | ย | ข | อ | ง | จ | ะ | ข | า | ย | ข | อ | ง

  00100011101110011101100111100100010110111001110110011110

  [$23: 35][$B9:185][$D9:217][$E4:228][$5B:91] [9D$:157][$9E:158]

  ห | ล | า | ก | ห | ล | า | ย |

  11110001101000111100011001100000 ← เติม 0 ให้เต็มไบท์

  [$F1:241][$A3:163][$C6:198][$60: 96]

char, bit, num

1 ไบท์,จำนวนรหัส

มีรหัสตามมา 12 ชุด x ชุดละ 3 ไบท์

4 ไบท์,จำนวนบิทรวมที่ต้องอ่านต่อ(นำไปถอดรหัส

- **ตัวอย่างการสร้าง Header ก่อนรับส่งข้อมูล (ค่าสมมุติ ขึ้นอยู่กับการออกแบบ)**

  [12][ก][4][8][ข][3][5][ค][4][2][ง][4][14][ล][4][4][น][4][3]

  [ย][3][3][ล][3][0][ห][4][15][อ][4][9][ะ][4][5][า][3][6][83]

- **ข้อมูลที่ส่งตามหลัง Header**

  [35][185][217][228][91][157][158][241][163][198][96] ← อ่านต่ออีก 83 บิท(11 ไบท์)

- **จำนวนบิทเฉลี่ยต่อตัวอักษร** = 83/24 = 3.45
- **อัตราส่วนการบีบอัด** = *83/(24*8)* = 43.22 %
- **Header = 1+12*3+4 = 41 ไบท์**