INFOPRESS

โครงสร้างข้อมูลและอัลกอริทึม + การประยุกต์ใช้กับ AI

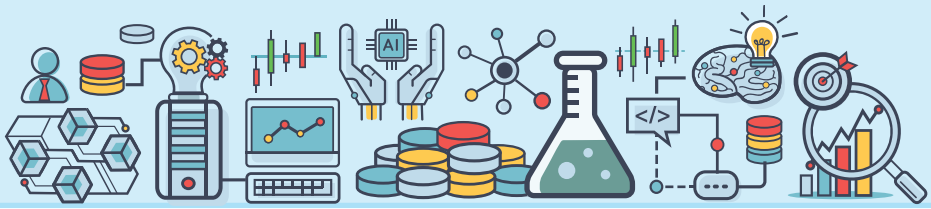
DATA STRUCTURE & ALGORITHM + AI

3rd Edition

- ครอบคลุมหลักการพื้นฐาน และการประยุกต์ใช้งานโครงสร้างข้อมูล อัลกอริทึม และใช้งานร่วมกับ AI
- ตัวอย่างการ Coding ด้วย ภาษา C และ Java พร้อมด้วยคำอธิบายอย่างละเอียด
- ใช้ประกอบการเรียนในระดับมหาวิทยาลัย, การวิจัย และการประยุกต์ใช้ในการทำงานจริง

ผศ.วิษณุ ช้างเนียม

ไฟล์ตัวอย่างภายในเล่ม
https://serazu.com/
9786164876248



สารบัญ

ตัวอย่างที่ 2.4 การวิเคราะห์เวลาที่ใช้ในการประมวลผลของโปรแกรมแบบวนซ้ำในการคำนวณหาแฟกทอเรียล

```
1  +Factorial(in n:int):int                    ← ถูกเรียกใช้ n ครั้ง
2     if (n == 0)                              ← ตรวจสอบเงื่อนไข n ครั้ง
3        return 1                              ← คืนค่า 1 ครั้ง
4     else return n * (Factorial(n-1))         ← เรียกใช้ตัวเอง n ครั้ง
```

อธิบายการวิเคราะห์เวลาที่ใช้ในการประมวลผล

บรรทัดที่ 1 ฟังก์ชันจะถูกเรียกใช้ตัวเองเท่ากับจำนวนการคำนวณแฟกทอเรียล n ครั้ง

บรรทัดที่ 2 ตรวจสอบว่า n = 0 หรือไม่ ในบรรทัดนี้ต้องตรวจสอบ n ครั้ง

บรรทัดที่ 3 ทำงาน 1 ครั้ง เพื่อคืนค่าเมื่อ n = 0

บรรทัดที่ 4 เรียกใช้งานตัวเองจำนวน n ครั้ง

ถ้ากำหนดให้ f(n) แทนประสิทธิภาพในการวิเคราะห์เวลาที่ใช้ในการประมวลผล และ n แทนจำนวนรอบในการคำนวณแฟกทอเรียล จะได้ว่าอัลกอริทึมนี้มีประสิทธิภาพ ดังนี้

$$f(n) = n + n + 1 + n = 3n + 1$$

อัตราการเติบโตของอัลกอริทึม (Algorithm Growth Rates)

การวัดประสิทธิภาพด้วยอัตราการเจริญเติบโตของอัลกอริทึม เป็นการวิเคราะห์อัลกอริทึมโดยใช้สัญลักษณ์มาเป็นตัวบ่งบอกว่าใช้เครื่องมือแบบไหนในการวัดประสิทธิภาพของอัลกอริทึม มีเครื่องมือที่ใช้ในการวัดประสิทธิภาพอัลกอริทึม ได้แก่ Big-O, Big-Omega, Big-Theta, Little-o และ Little-omega เป็นต้น

อัตราการเติบโต Big-O

อัตราการเติบโต Big-O เป็นการวัดประสิทธิภาพเชิงเวลาที่ใช้ในการประมวลผลของอัลกอริทึม ซึ่งเป็น**ฟังก์ชันเวลาขอบเขตบนที่ใช้ในการประมวลผล (Asymptotic Upper Bound)** หมายถึง กรณีที่ใช้เวลาประมวลผลนานที่สุดที่โปรแกรมประมวลผล

อัตราการเติบโต Big-O ใช้สัญลักษณ์เป็นตัวโอใหญ่ (O) กำกับในการแสดงการวัดประสิทธิภาพ

ดังนั้น การใช้สัญลักษณ์ Big-O กำกับในการวัดประสิทธิภาพเชิงเวลาของอัลกอริทึมจะเป็นการรับรองว่าเวลาที่ใช้ประมวลผลอัลกอริทึมจะไม่มากกว่าเวลาที่คำนวณได้จากฟังก์ชัน Big-O ยกตัวอย่างเช่น O(n) หมายถึง **อัลกอริทึม**นี้จะใช้เวลาในการประมวลผลน้อยกว่าหรือเท่ากับ n ($\leq$ n) เสมอ

ลิงค์ลิสต์ทิศทางเดียวแบบวงกลมได้เปลี่ยนการเชื่อมโยงโหนดสุดท้ายของลิงค์ลิสต์จาก **null** เป็นการอ้างอิงไปยังโหนดแรกของลิงค์ลิสต์แทน เมื่อพิจารณาจากลักษณะการอ้างอิงข้อมูลแบบนี้ จึงเรียกโครงสร้างลิงค์ลิสต์แบบนี้ว่า **"ลิงค์ลิสต์แบบวงกลม" (Circular Linked-list)**

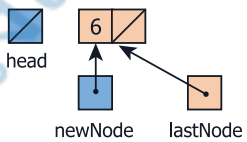
การจัดการลิงค์ลิสต์ทิศทางเดียวแบบวงกลม

การจัดการลิงค์ลิสต์ทิศทางเดียวแบบวงกลม มีหลักการทำงานเหมือนกับการจัดการลิงค์ลิสต์ทิศทางเดียว แต่เพิ่มตัวแปร lastNode ทำหน้าอ้างอิงตำแหน่งสุดท้ายของลิงค์ลิสต์ เพื่อใช้สำหรับกำหนดการวนกลับไปอ้างอิงตำแหน่งส่วนหัวของลิงค์ลิสต์ ดังแสดงการสร้างลิงค์ลิสต์ทิศทางเดียวแบบวงกลม ดังนี้

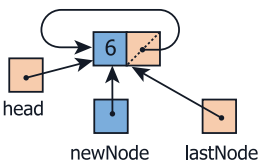
ตัวอย่างที่ 4.14 การสร้างลิงค์ลิสต์ทิศทางเดียวแบบวงกลม

(a) [Java] `Node head = new Node();`
[C] `struct Node *head;`

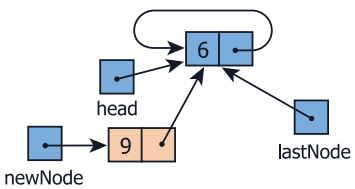
(b) [Java]
```
Node newNode = new Node(6);
Node lastNode = newNode;
```
[C]
```
struct Node *newNode;
newNode = createNode(6, NULL);
struct Node *lastNode = newNode;
```



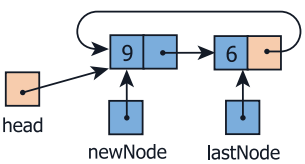
(c) [Java]
```
head = newNode;
lastNode.setNext(newNode);
```
[C]
```
head = newNode;
setNext(lastNode, newNode);
```

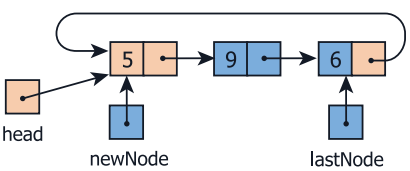


(d) [Java] `newNode = new Node(9, head);`
[C] `newNode = createNode(9, head);`



(e) Java]
```
lastNode.setNext(newNode);
head = newNode;
```
[C]
```
setNext(lastNode, newNode);
head = newNode;
```





ตัวอย่างที่ 5.11 ลำดับขั้นตอนการคำนวณทางคณิตศาสตร์จากรูปแบบ Postfix

```
Postfix: abcd*-e/+
a = 20, b = 5, c = 3, d = 5, e = 4
Result: 17.5
```

postfix	calculator action	stack
a	push a	a
b	push b	a b
c	push c	a b c
d	push d	a b c d
*	Op2 = pop stack (d = 5)	a b c
	Op1 = pop stack (c = 3)	a b
	f = Op1 * Op2 (f = 15)	a b
	push f	a b f
-	Op2 = pop stack (f = 15)	a b
	Op1 = pop stack (b = 5)	a
	f = Op1 - Op2 (f = -10)	a
	push f	a f
e	push e	a f e
/	Op2 = pop stack (e = 4)	a f
	Op1 = pop stack (f = -10)	a
	f = Op1 / Op2 (f = -2.5)	a
push f	a f	
+	Op2 = pop stack (f = -2.5)	a
	Op1 = pop stack (a = 20)	
	f = Op1 + Op2 (f = 17.5)	
	push f	f

อธิบายการทำงาน

เมื่อ ch เป็นตัวอักษร นำข้อมูลตัวอักษรเก็บไว้ในสแต็ก

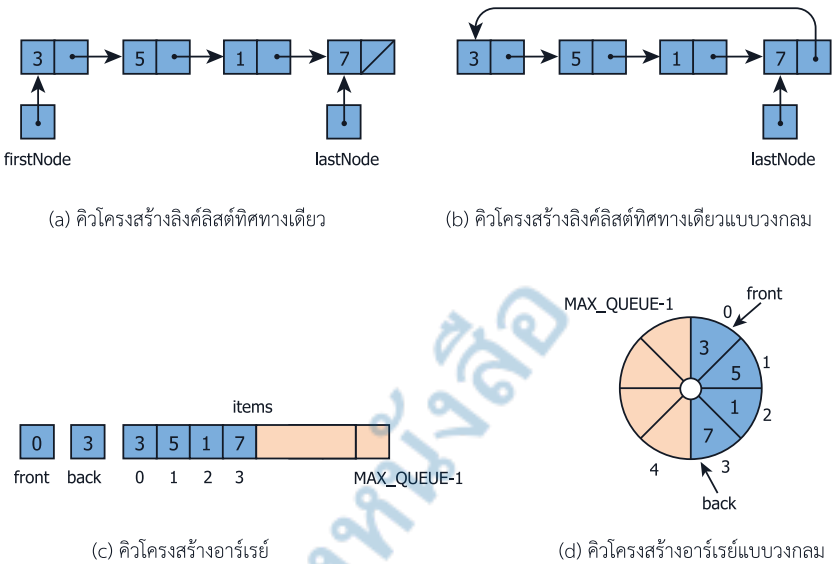
เมื่อ ch เป็นเครื่องหมายทางคณิตศาสตร์ นำข้อมูลตัวอักษรออกจากสแต็กเก็บไว้ใน Op2 และนำข้อมูลตัวอักษรออกจากสแต็กเก็บไว้ใน Op1 กระทำตามเครื่องหมายทางคณิตศาสตร์ ด้วยการนำข้อมูลตัวเลขในตัวอักษรมากระทำทางคณิตและเก็บไว้ในตัวแปร f (f = Op1 operator Op2) นำตัวอักษร f เก็บลงในสแต็ก

จากขั้นตอนการคำนวณทางคณิตศาสตร์ด้วยรูปแบบของ Postfix สามารถนำมาเขียนเป็นอัลกอริทึมภาษา Java และภาษา C ได้ดังนี้

เครื่องมือที่ใช้สร้างคิว

เครื่องมือหรือรูปแบบการเก็บข้อมูลที่นำมาสร้างคิวมี 2 เครื่องมือ คือ อาร์เรย์ และลิงค์ลิสต์ ดังแสดงโครงสร้างคิวจากโครงสร้างอาร์เรย์ และลิงค์ลิสต์ในรูปที่ 6-1

รูปที่ 6-1
แสดงเครื่องมือที่ใช้สร้างคิว



(a) คิวโครงสร้างลิงค์ลิสต์ทิศทางเดียว

(b) คิวโครงสร้างลิงค์ลิสต์ทิศทางเดียวแบบวงกลม

(c) คิวโครงสร้างอาร์เรย์

(d) คิวโครงสร้างอาร์เรย์แบบวงกลม

การสร้างคิวด้วยโครงสร้างลิงค์ลิสต์

โครงสร้างลิงค์ลิสต์ เป็นรูปแบบการเก็บข้อมูลที่สามารถนำมาสร้างคิวได้อย่างมีประสิทธิภาพในการใช้พื้นที่หน่วยความจำ และสะดวกในการจัดการเกี่ยวกับการนำข้อมูลเข้าและออกจากคิว

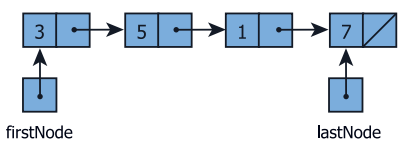
การสร้างคิวด้วยโครงสร้างลิงค์ลิสต์ สามารถใช้โครงสร้างลิงค์ลิสต์ทิศทางเดียว หรือลิงค์ลิสต์ทิศทางเดียวแบบวงกลมมาเป็นเครื่องมือในการสร้างคิวก็ได้ โดยมีรายละเอียดดังต่อไปนี้

การสร้างคิวด้วยโครงสร้างลิงค์ลิสต์ทิศทางเดียว

การนำลิงค์ลิสต์ทิศทางเดียวมาสร้างเป็นคิว มีตัวแปรทำหน้าที่ในการอ้างอิงตำแหน่งโหนดในลิงค์ลิสต์อยู่สองตัวแปร และแสดงคิวโครงสร้างลิงค์ลิสต์ทิศทางเดียวในรูปที่ 6-2

- firstNode อ้างอิงตำแหน่งโหนดข้อมูลที่เข้ามาในคิวก่อน
- lastNode อ้างอิงตำแหน่งโหนดข้อมูลที่เข้ามาในคิวในลำดับสุดท้าย

รูปที่ 6-2
คิวโครงสร้างลิงค์ลิสต์ทิศทางเดียว



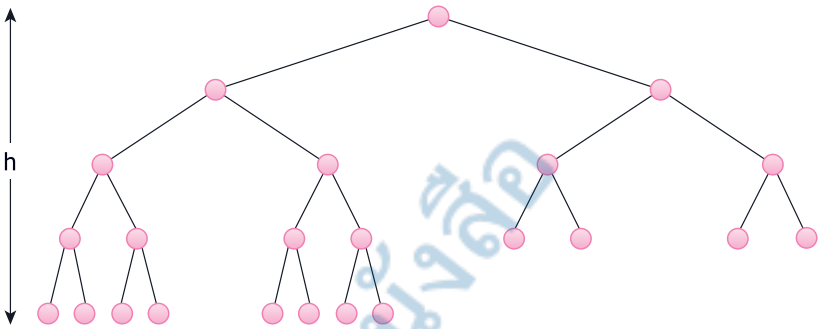
คุณสมบัติไบนารีทรีเต็ม (Full Binary Tree) จะมีข้อกำหนด ดังนี้

- ถ้าทรีว่างจะเป็นไบนารีทรีเต็ม เมื่อความสูงของทรีมีค่าเท่ากับศูนย์ (h = 0)
- ถ้าทรีไม่ว่าง คือ h > 0 และที่ตำแหน่งความสูง h – 1 มีโหนดครบทุกโหนดจึงจะเป็นไบนารีทรีเต็ม

ไบนารีทรีแบบสมบูรณ์ (Complete Binary Tree) จะมีข้อกำหนด ดังนี้

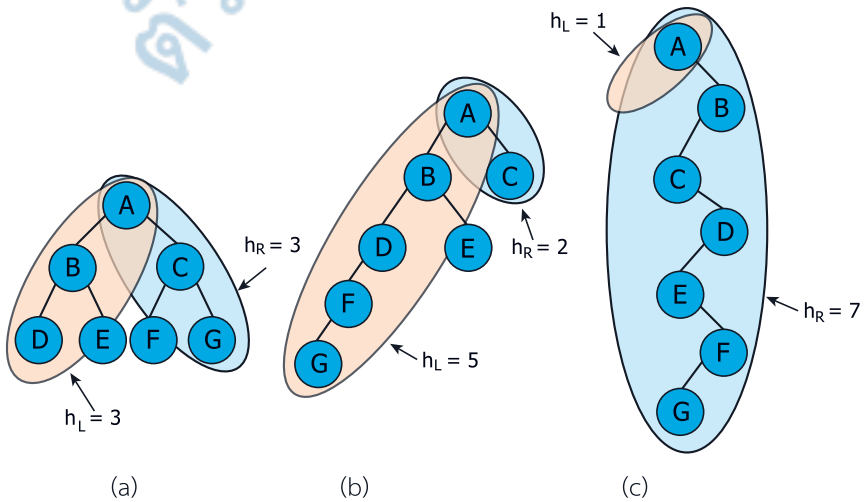
- ที่ตำแหน่งความสูง h – 1 จะต้องมีโหนดเต็มทุกตำแหน่ง
- การเพิ่มโหนดเข้าไปในทรีที่ตำแหน่งความสูง h ต้องเพิ่มโหนดในทรีจากซ้ายไปขวา ดังแสดงโครงสร้างไบนารีทรีแบบสมบูรณ์ในรูปที่ 7-6

รูปที่ 7-6 โครงสร้างไบนารีทรีแบบสมบูรณ์

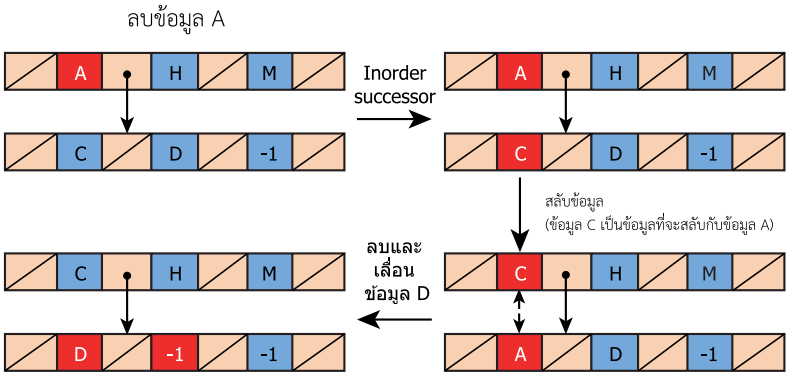


ความสูงสมดุล (Height Balanced) หรือ**ทรีแบบสมดุล** เมื่อความสูงของโหนดย่อยทางขวากับความสูงของโหนดย่อยทางซ้ายจะต้องมีความแตกต่างของความสูงไม่เกิน 1 จึงจะถือว่าเป็นทรีแบบสมดุล และเมื่อพิจารณาทรีในรูปที่ 7-5 นำมากำหนดความสูงด้านซ้าย และด้านขวา ดังรูปที่ 7-7 และพิจารณาได้ว่า ไบนารีทรีรูปที่ 7-7 (b) และ 7-7 (c) เป็นทรีแบบไม่สมดุล เพราะความสูงทรีย่อยด้ายซ้ายและด้านขวามีความสูงต่างกันเกิน 1 แต่ไบนารีทรีรูปที่ 7-7 (a) เป็นทรีที่สมดุลและเป็นทรีแบบสมบูรณ์ด้วย

รูปที่ 7-7 แสดงความสูงทรีย่อยทางซ้ายและทางขวา

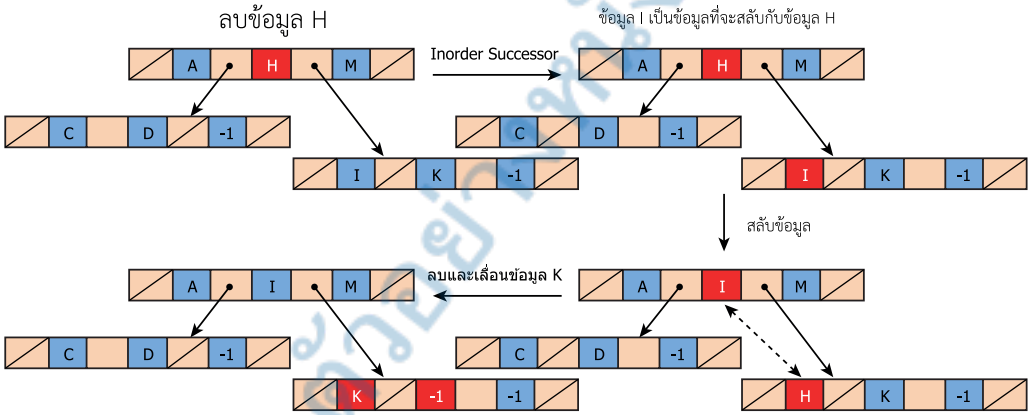


2. กรณีโหนดลูกอยู่ด้านขวาของข้อมูลโหนดที่ต้องการลบ ในกรณีนี้จะใช้หลักการ Inorder Successor ในการหาตำแหน่งข้อมูลมาแทนที่ข้อมูลโหนดแม่ที่ต้องการลบ



❖ **กรณีข้อมูลที่ต้องการลบมีโหนดลูกอยู่ 2 โหนด**

การลบข้อมูลในกรณีนี้จะใช้หลักการ Inorder Successor ในการหาตำแหน่งข้อมูลมาแทนที่ข้อมูลที่ต้องการลบ



จากรูปแบบในการลบข้อมูลใน K-ary ทรี นำมาเขียนเป็น โค้ดรหัสเทียมได้ดังนี้

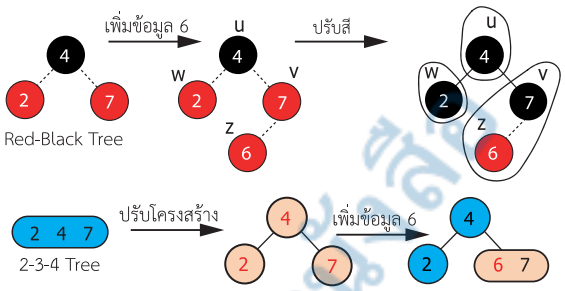
กรณีที่ 2 : w เป็นสีแดง

ในกรณีโหนด w เป็นโหนดสีแดง ถ้าใน 2-3-4 Trees แสดงว่าจะมีข้อมูล 4 ค่าในหนึ่งโหนด ซึ่งผิดกฎข้อที่ 1 ของ 2-3-4 Trees การปรับโครงสร้างในกรณีของ Red-Black Tree จะใช้หลักการในการปรับสีของโหนดเพื่อทำให้ทรีสมดุล ตามขั้นตอนดังนี้

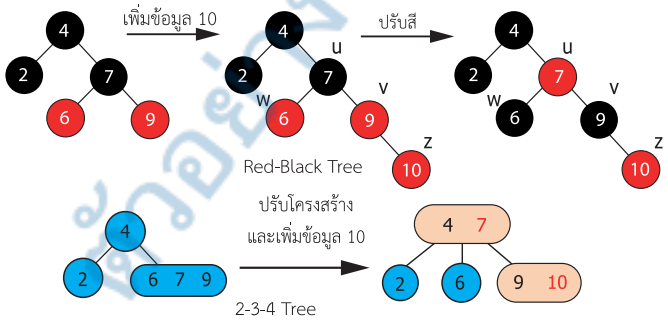
- เปลี่ยนสีโหนดแม่ คือ โหนด v และโหนดระดับพี่น้อง คือ โหนด w ให้เป็นโหนดสีดำ
- เปลี่ยนสีโหนดแม่ของโหนด v คือ โหนด u เปลี่ยนเป็นโหนดสีแดง ยกเว้นถ้าโหนด u เป็นโหนดรากให้เป็นโหนดสีดำเหมือนเดิม

โดยแสดงการเปลี่ยนสีโหนดใน Red-Black Tree ให้สมดุลดังรูปที่ 8-49

รูปที่ 8-49 แสดงการปรับสีโหนดในกรณีโหนด w เป็นโหนดสีแดง

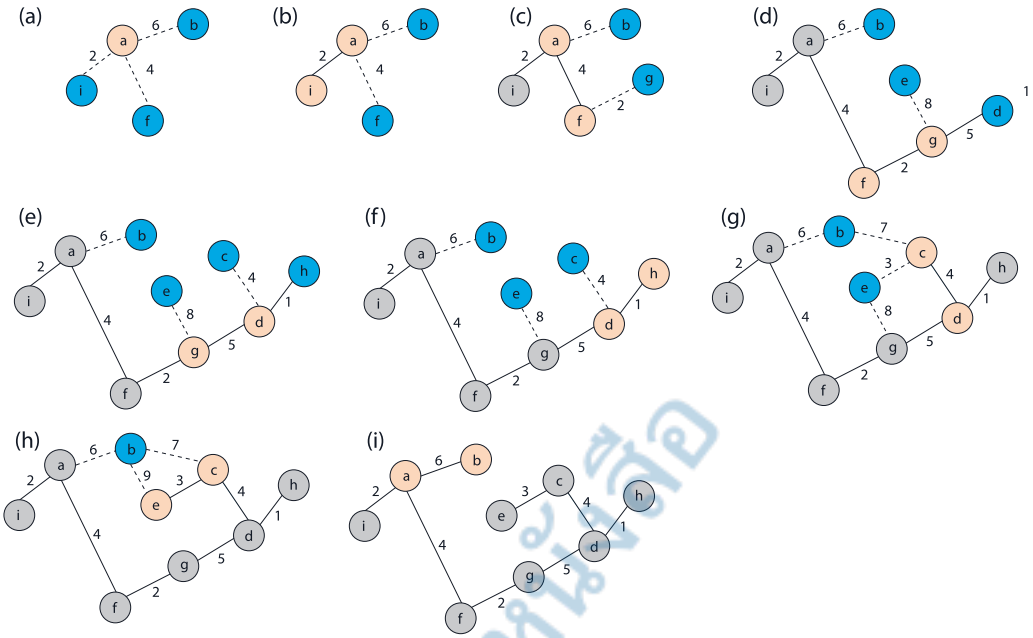


a) เพิ่มโหนดในกรณีโหนด w เป็นโหนดสีแดง และโหนด u เป็นโหนดราก



b) เพิ่มโหนดในกรณีโหนด w เป็นโหนดสีแดง และโหนด u ไม่ใช่โหนดราก

ตัวอย่างที่ 10.10 การหาเส้นทางที่สั้นที่สุดด้วยอัลกอริทึม Prim



อธิบายการทำงาน

ขั้นตอน a กำหนดให้โหนด a เป็นโหนดเริ่มต้นในการท่องเข้าไปในกราฟ และกำหนดให้โหนด a เป็นโหนดที่ถูกท่องเข้าไปแล้ว

ขั้นตอน b โหนด a มีโหนดประชิดที่ยังไม่เคยถูกท่องเข้าไป คือ โหนด b, f และ i จากทั้ง 3 เส้นทาง เส้นทาง (a, i) มีน้ำหนักเส้นทางที่น้อยที่สุดเมื่อเทียบทั้ง 3 เส้นทาง ดังนั้น กำหนดให้โหนด i เป็นโหนดที่ถูกท่องเข้าไปแล้ว

ขั้นตอน c เนื่องจากโหนด i ไม่มีโหนดประชิด ดังนั้น ต้องถอยกลับไปยังโหนดต้นทาง คือ โหนด a และหาโหนดประชิดที่ยังไม่ได้ถูกท่องเข้าไป คือ โหนด f และ b เส้นทางระหว่างโหนด (a, f) มีค่าน้ำหนักน้อยกว่าเส้นทาง (a, b) ดังนั้น เลือกเส้นทาง (a, f) และกำหนดให้โหนด f เป็นโหนดที่ถูกท่องเข้าไปแล้ว

ขั้นตอน d โหนดประชิดของโหนด f คือ โหนด g เพียงโหนดเดียว และเป็นโหนดที่ยังไม่ถูกท่องเข้าไป ดังนั้น เลือกเส้นทาง (f, g) และกำหนดให้โหนด g เป็นโหนดที่ถูกท่องเข้าไปแล้ว

ขั้นตอน e โหนด g มีโหนดประชิด คือ โหนด d และ e เส้นทางระหว่างโหนด (g, d) มีน้ำหนักน้อยกว่า (g, e) ดังนั้น เลือกเส้นทาง (g, d) และกำหนดให้โหนด d เป็นโหนดที่ถูกท่องเข้าไปแล้ว

วิเคราะห์หาประสิทธิภาพการจัดเรียงข้อมูลแบบ Bubble Sort

การจัดเรียงข้อมูลแบบ Bubble Sort ในรอบแรกจะเปรียบเทียบข้อมูลทั้งหมด n - 1 ครั้ง และจะต้องสลับตำแหน่งข้อมูลไม่เกินกว่า n - 1 ครั้ง ส่วนในรอบที่สองเปรียบเทียบข้อมูล n - 2 ครั้ง และจะต้องสลับตำแหน่งข้อมูลไม่เกินกว่า n - 2 ครั้ง ดังนั้น แสดงว่าการวนรอบที่ i จะต้องเปรียบเทียบข้อมูล n - i ครั้ง และจะต้องสลับตำแหน่งข้อมูลไม่เกินกว่า n - i ครั้ง ซึ่งสามารถหาค่า Worst-Case ของการจัดเรียงข้อมูลแบบ Bubble Sort ได้ดังนี้

1. ในลูป for ในบรรทัดที่ 2-8 วนลูปเท่ากับ n ครั้ง
2. ในลูป for ในบรรทัดที่ 3-7 วนลูปเท่ากับ $\frac{n-1}{2}$ ครั้ง
3. ในบรรทัดที่ 4-5 ทำคำสั่งเท่ากับ 2 คำสั่ง

ดังนั้น Worst-Case เท่ากับ $= \sum_{i=n-1}^{1}\sum_{i=1}^{n-i}2 = \sum_{i=1}^{n}2\left(\frac{n-1}{2}\right) = 2n\left(\frac{n-1}{2}\right) = n^2-n$

ทำให้ **Worst-Case ของการจัดเรียงข้อมูลแบบ Bubble Sort เท่ากับ $O(n^2)$**

ส่วน Best-Case ในการจัดเรียงข้อมูลแบบ Bubble Sort คือ จัดเรียงข้อมูลเพียงรอบเดียว คือ เปรียบเทียบข้อมูลเพียง n - 1 รอบ และไม่มีการสลับตำแหน่งข้อมูล **ทำให้ Best-Case ของการจัดเรียงข้อมูลแบบ Bubble Sort เท่ากับ O(n)**

โดยแสดงอัลกอริทึมการจัดเรียงข้อมูลแบบ Bubble Sort ได้ดังนี้

Java

```
public static Comparable[] bubbleSort(Comparable[] theArray,int n){
   for(int pass=1;pass<n;pass++){
        for(int index=0; index<n-pass; index++){
           int nextIndex = index+1;
           if(theArray[index].compareTo(theArray[nextIndex])>0){
                Comparable temp = theArray[index];
                theArray[index] = theArray[nextIndex];
                theArray[nextIndex] = temp;
            }
        }
   }
   return theArray;
}
```

C

```
int* bubbleSort(int theArray[],int n){
   for(int pass=1;pass<n;pass++){
       for(int index=0;index<n-pass;index++){
           int nextIndex = index+1;
           if(theArray[index]> theArray[nextIndex]){
             int temp = theArray[index];
             theArray[index] = theArray[nextIndex];
             theArray[nextIndex] = temp;
          }
       }
   }
   return theArray;
}
```

CHAPTER

12

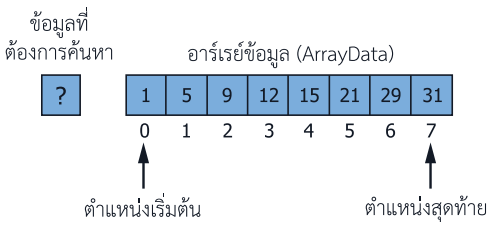
การค้นหาข้อมูล (Searching)

การค้นหาข้อมูล คือ การนำข้อมูลที่ถูกจัดเรียงแล้วมาค้นหาข้อมูลที่ต้องการค้นหา ก่อนที่จะค้นหาข้อมูลได้ต้องจัดเรียงข้อมูลที่ต้องการค้นหาให้เรียบร้อย เนื่องจากข้อมูลที่กระจัดกระจายไม่เป็นลำดับ เมื่อค้นหาข้อมูลต้องวนลูปตั้งแต่ข้อมูลตำแหน่งแรกถึงตำแหน่งสุดท้ายทุกครั้ง ไม่สามารถหยุดการค้นหาได้ เนื่องจากข้อมูลที่ต้องการค้นหาอาจอยู่ในตำแหน่งสุดท้ายก็ได้

ดังนั้น ภายในบทนี้จะแสดงการค้นหาข้อมูลเฉพาะข้อมูลที่เป็นตัวเลขเท่านั้น ส่วนการค้นหาที่เป็นข้อมูลตัวอักษรจะกล่าวถึงในบทที่ 14

การค้นหาข้อมูลตัวเลขแบบลำดับ (Sequential Search หรือ Linear Search)

การค้นหาข้อมูลแบบลำดับ เป็นการค้นหาข้อมูลที่เป็นแบบตัวเลขในอาร์เรย์ขนาด n ข้อมูล ดังแสดงอาร์เรย์ข้อมูลเพื่อใช้สำหรับการค้นหาในรูปที่ 12-1 และมีลำดับขั้นตอนการค้นหาข้อมูลแบบลำดับ ดังนี้

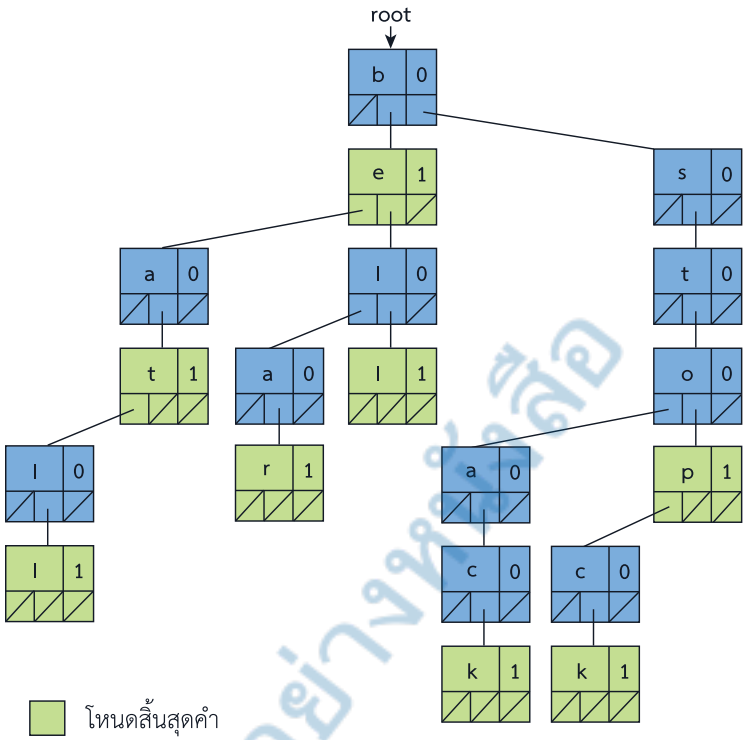


รูปที่ 12-1 อาร์เรย์ข้อมูลเพื่อใช้สำหรับการค้นหาข้อมูล

ตัวอย่างที่ 14.12 การเก็บคำด้วยโครงสร้าง Ternary Search Tree

ข้อความ : bell, be, bear, bat, ball, stop, stock และ stack

โครงสร้าง Ternary Search Tree:



การเพิ่มคำใน Ternary Search Tree

การเพิ่มคำใน Ternary Search Tree ถ้าทรีว่างเปล่า จะเพิ่มคำในทรีแล้วกำหนดให้ตัวอักษรตัวแรกของคำเป็นตำแหน่งโหนดราก เมื่อมีคำใหม่เข้ามาเพิ่มจะนำตัวอักษรตัวแรกของคำใหม่มาเปรียบเทียบกับตัวอักษรในตำแหน่งโหนดแรกตามรูปแบบ ดังนี้

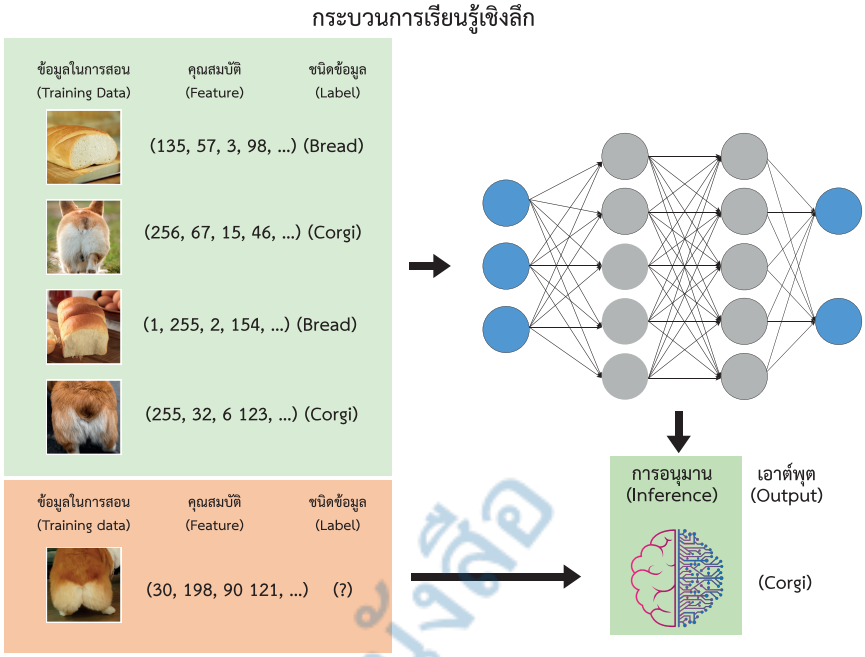
1. ถ้าตัวอักษรไม่เหมือนกัน จะเปรียบเทียบตัวอักษรของคำที่เพิ่มกับคำที่มีอยู่ในโครงสร้างทรีมีค่ามากกว่า หรือน้อยกว่า ดังนี้

   1.1 ถ้าน้อยกว่าให้ท่องไปทางซ้ายของโหนดเปรียบเทียบ

   1.2 ถ้ามากกว่าให้ท่องไปทางขวาของโหนดเปรียบเทียบ

   เมื่อท่องเข้าไปแล้วไม่มีโหนดลูก สามารถสร้างโหนดใหม่ตามโครงสร้าง TernaryNode แล้วเชื่อมเข้ากับโหนดปัจจุบันที่เปรียบเทียบอยู่ตามตำแหน่งโหนดลูกทางซ้ายหรือทางขวา

รูปที่ 15-7
แสดงการเรียนรู้เชิงลึกในการแยกแยะวัตถุ



หลักการทำงานวิเคราะห์ข้อมูลอินพุตที่เข้ามาในการเรียนรู้เชิงลึก จะเริ่มจากนำรูปภาพที่เข้ามาทางชั้นอินพุตมาคำนวณหาความสอดคล้องกับการจัดหมวดหมู่ของวัตถุที่เรียนรู้มาแล้ว ว่ามีค่าความสอดคล้องกับวัตถุที่ศึกษามาแล้วว่ามีความสอดคล้องกับวัตถุนั้นมากน้อยเพียงใด และส่งค่าความสอดคล้องกับวัตถุส่งต่อไปยังชั้นถัดๆ ไปจนถึงชั้นเอาต์พุต เพื่อบ่งบอกว่าข้อมูลอินพุตที่เข้ามาเป็นชนิดอะไร เครือข่ายประสาทเทียมมีความสำเร็จจากข้อมูลที่ฝึกฝนเมื่อค่าของน้ำหนักให้ผลลัพธ์ที่ใกล้เคียงกับความเป็นจริงมากที่สุด

การเรียนรู้เชิงลึกนำไปใช้งานจริงหลากหลายรูปแบบเพื่อใช้ในการเรียนรู้จดจำ ที่เห็นได้ชัดที่นำไปใช้งานและประสบความสำเร็จ คือ การเปลี่ยนภาษาใน Google Translate ที่มีความสามารถในการเปลี่ยนคำหรือประโยคให้เป็นภาษาต่างๆ ได้มากกว่า 100 ภาษา

ปัญญาประดิษฐ์ (Artificial Intelligence)

ปัญญาประดิษฐ์ (Artificial Intelligence) หรือ **AI** เป็นเพียงอัลกอริทึม โค้ดคำสั่ง หรือเทคนิคที่ช่วยให้เครื่องจักรสามารถเลียนแบบ พัฒนา และเรียนรู้พฤติกรรมของมนุษย์ได้ ในโลกปัจจุบัน AI เป็นรูปแบบการทำงานที่มีประสิทธิภาพที่สามารถนำมาใช้งาน หรือช่วยงานในชีวิตประจำวันของมนุษย์ได้อย่างมีประสิทธิภาพ เช่นเดียวกับที่มนุษย์ทำ คือ การเรียนรู้ การวางแผน การใช้เหตุผล การตัดสินใจ และการแก้ปัญหา

AI มีรูปแบบในการพัฒนาอยู่ในช่วงที่ต้องฝึกฝนให้ AI เรียนรู้และฝึกฝนให้ทำสิ่งใดสิ่งหนึ่งเท่านั้น ดังรูปแบบการพัฒนาโปรแกรม AI ให้มีทักษะดังต่อไปนี้

เทคนิคการวิเคราะห์ข้อมูลทางสถิติ (Statistical Data Analysis Techniques)

เทคนิคพื้นฐานเพื่อใช้ในการวิเคราะห์ข้อมูลทางสถิติ ได้แก่ การหาค่าเฉลี่ย ค่ามัธยฐาน ข้อมูลที่ซ้ำกันมากที่สุด ค่าเบี่ยงเบนมาตรฐาน โดยมีรูปแบบการทำงาน ดังนี้

คำนวณค่าเฉลี่ย (Calculating the mean)

การคำนวณค่าเฉลี่ยใช้วิธีการบวกค่าทั้งหมดในรายการ แล้วหารด้วยจำนวนข้อมูลทั้งหมดในรายการข้อมูล เทคนิคนี้มีประโยชน์ในการกำหนดแนวโน้มของข้อมูล หรือนำข้อมูลค่ากลางไปใช้สำหรับเติมข้อมูลที่หายไป

ตัวอย่างที่ 16.6 การหาค่าเฉลี่ย

```
1   public class MeanDataset {
2       public static void main(String[] args) {
3           double[] testData = {11.5, 12.7, 15.2, 13.0, 25.1, 34.3, 17.9, 22.5, 28.8,
    27.9};
4           double total = 0;
5           System.out.print("ข้อมูล: ");
6           for (double element : testData) {
7               total += element;
8               System.out.print(" "+element);
9           }
10          System.out.println();
11          double mean = total / testData.length;
12          System.out.println("ค่าเฉลี่ย: "+ mean);
13      }
14  }
```

อธิบายการทำงาน

บรรทัดที่ 1	ประกาศคลาสทำหน้าที่หาค่าเฉลี่ย
บรรทัดที่ 2	ประกาศเมธอดหลักในการทำงาน
บรรทัดที่ 3	ประกาศชุดข้อมูลที่ต้องการหาค่าเฉลี่ย
บรรทัดที่ 4	ประกาศตัวแปร total ทำหน้าที่เก็บข้อมูลผลรวมทั้งหมดของชุดข้อมูล
บรรทัดที่ 5	แสดงข้อความที่ส่วนแสดงผลเพื่อบ่งบอกว่าข้อมูลที่แสดงต่อไปเป็นส่วนข้อมูล
บรรทัดที่ 6-9	วนลูปเท่ากับข้อมูลในชุดข้อมูลและบวกค่าในชุดข้อมูลทีละค่าพร้อมแสดงผลข้อมูลที่ส่วนแสดงผล
บรรทัดที่ 10	ขึ้นบรรทัดใหม่ในส่วนแสดงผล
บรรทัดที่ 11	หาค่าเฉลี่ยด้วยการนำผลบวกข้อมูลทั้งหมดที่เก็บไว้ในตัวแปร total มาหารด้วยจำนวนข้อมูลทั้งหมด แล้วเป็นผลไว้ในตัวแปร mean
บรรทัดที่ 12	แสดงค่าเฉลี่ยที่ส่วนแสดงผล

ผลการทำงาน

ข้อมูล : 11.5 12.7 15.2 13.0 25.1 34.3 17.9 22.5 28.8 27.9
ค่าเฉลี่ย : 20.0
ค่าเบี่ยงเบนมาตรฐาน : 7.54983443527075

การวิเคราะห์ทางปัญญาประดิษฐ์ (Artificial Intelligence Analysis)

การวิเคราะห์ข้อมูลทางปัญญาประดิษฐ์ มีเครื่องมือที่ใช้งาน ได้แก่ การเรียนรู้ของเครื่องจักร เครือข่ายประสาทเทียม การเรียนรู้เชิงลึก

การเรียนรู้ของเครื่องจักร (Machine Learning)

การเรียนรู้ของเครื่องจักรเป็นหัวข้อกว้างๆ ภาษาจาวามีไลบรารีสนับสนุนมากมาย โดยทั่วไปไลบรารีจะเป็นเครื่องมือที่ช่วยพัฒนาเทคนิคต่างๆ ให้สะดวกในการพัฒนาระบบการเรียนรู้ของเครื่องจักร โดยไม่ต้องเรียนรู้หลักการทำงานที่ซับซ้อนมาก ปกติแล้วโมเดลที่ถูกสร้างขึ้นจะเป็นโมเดลที่ถูกสร้างขึ้นเพื่อแก้ไขปัญหาใดปัญหาหนึ่ง โดยเริ่มจากการสอนให้เครื่องจักรเรียนรู้ตัวอย่างข้อมูลที่ต้องการแก้ไขปัญหา แล้วนำโมเดลที่สอนเสร็จแล้วไปทดสอบประสิทธิภาพของโมเดล ด้วยการใช้ข้อมูลทดสอบ เมื่อโมเดลมีประสิทธิภาพในการแก้ไขปัญหาได้อย่างเหมาะสมก็นำโมเดลไปใช้งานกับข้อมูลใหม่เพื่อการคาดการณ์ผลลัพธ์

ตัวอย่างการใช้งานการเรียนรู้ของเครื่องจักรเพื่อเป็นโมเดลจัดการเกี่ยวกับการซื้อสินค้าของลูกค้า ด้วยการนำข้อมูลการซื้อสินค้าของลูกค้ามาเป็นข้อมูลสำหรับสอนเครื่องจักร เพื่อสร้างโมเดลการเรียนรู้ เพื่อใช้เป็นโมเดลสำหรับการคาดการณ์ลูกค้าที่มีรูปแบบการซื้อที่คล้ายคลึงกัน เพื่อนำเสนอสินค้าที่เหมาะสมให้กับลูกค้า หรือนำผลการคาดการณ์ไปใช้ในจัดการทำข้อเสนอพิเศษสำหรับลูกค้า หรือสร้างการส่งเสริมการขายเพื่อดึงดูดลูกค้าให้กลับมาซื้อสินค้าใหม่ หรือสร้างรูปแบบการจัดสินค้าให้มีความสะดวกในการเลือกซื้อสินค้า จากการวิเคราะห์ข้อมูลการซื้อสินค้าจากสินค้าชิ้น A แล้วจะต้องซื้อสินค้าชิ้น B ดังนั้น ทางร้านก็สามารถจัดสินค้าภายในร้านให้สินค้า A และ B อยู่ใกล้ๆ กันเพื่อเพิ่มความสะดวกในการซื้อสินค้าของลูกค้า เป็นต้น

การเรียนรู้ของเครื่องจักรแบบการเรียนรู้โดยมีผู้สอน

การเรียนรู้ของเครื่องจักรที่ช่วยในการตัดสินใจแบบการเรียนรู้โดยมีผู้สอน ได้แก่ ต้นไม้ตัดสินใจ (Decision Tree) เป็นแบบจำลองที่ใช้ในการคาดการณ์ตามรูปแบบการตัดสินใจของมนุษย์ ซึ่งการตัดสินใจแต่ละครั้งจะนำเอาปัจจัยต่างๆ มาเป็นข้อมูลประกอบการตัดสินใจในแต่ละครั้ง ตัวอย่างเช่น ถ้ามีเพื่อนชวนไปกินก๋วยเตี๋ยวกันไหม คนที่ถูกชวนมักจะคิดก่อนว่า ก๋วยเตี๋ยวที่จะไปอร่อยไหม ราคาแพงไหม แล้วจึงตัดสินใจว่าจะไปหรือไม่

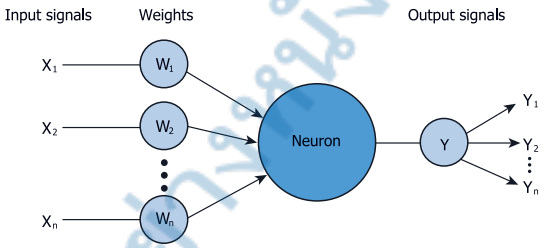
เครือข่ายประสาทเทียม (Artificial Neural Network)

ในการสร้างโมเดลเครือข่ายประสาทเทียมจะต้องมีสองโหมดในการทำงาน คือ

- **Training Mode** เป็นขั้นตอนของสอนให้เซลล์ประสาทเทียมได้รู้จักการจัดการ เมื่อมีข้อมูลอินพุตมีรูปแบบนี้ และผลของข้อมูลที่เป็นอินพุตเป็นอย่างไรเพื่อให้เซลล์ประสาทเทียมได้ศึกษาวิธีในการแก้ไขปัญหา
- **Testing Mode** เป็นขั้นตอนของการทดสอบโมเดลที่ถูกสอนตามโครงสร้างประสาทเทียมเรียบร้อยมาทดสอบด้วยข้อมูลอินพุตที่ต้องการทดสอบว่าผลการเรียนรู้ของโมเดลมีความถูกต้องหรือไม่

ชุดข้อมูลที่มีอยู่จะถูกแยกออกเป็นสองส่วน คือ ส่วนชุดข้อมูลที่มีจำนวนชุดข้อมูลมากกว่าอีกชุดหนึ่งจะถูกใช้สำหรับสอนเครือข่ายประสาทเทียม ส่วนชุดที่สองที่มีจำนวนข้อมูลน้อยกว่าจะใช้สำหรับการทดสอบโมเดลประสาทเทียมที่ถูกสร้างขึ้นมา

สัญญาณข้อมูลอินพุตเป็นสัญญาณข้อมูลดิบที่ยังไม่ได้ถูกปรับแต่ง หรือเป็นสัญญาณเอาต์พุตของเซลล์ประสาทอื่นที่รับเข้ามาในเซลล์ประสาทที่ทำงานอยู่ สัญญาณเอาต์พุตสามารถเป็นทั้งผลสรุปสุดท้ายในการแก้ไขปัญหา หรือเป็นอินพุตให้กับเซลล์ประสาทอื่นต่อไป ดังแสดงโครงสร้างเซลล์ประสาทในรูปที่ 16-2



รูปที่ 16-2 แสดงโครงสร้างของเซลล์ประสาทหนึ่งเซลล์

ผลเอาต์พุตของเซลล์ประสาท คำนวณได้จากผลรวมระหว่างค่าน้ำหนักอินพุตคูณกับค่าอินพุต แล้วนำมาเปรียบเทียบผลกับค่าที่กำหนดไว้ (Threshold Value) คือ θ ถ้าผลรวมน้อยกว่าค่าที่กำหนดไว้ ผลของเซลล์ประสาทเอาต์พุตจะมีค่าเท่ากับ -1 แต่ถ้าผลรวมมากกว่าหรือเท่ากับค่าที่กำหนดไว้ ผลของเซลล์ประสาทเอาต์พุตจะมีค่าเท่ากับ +1

จากหลักการที่กล่าวก่อนหน้านำมาเขียนให้อยู่ในรูปแบบของ **ฟังก์ชันการตอบสนอง (Activation Function)** ได้ดังนี้

$$X = \sum_{i=1}^{n} x_i w_i$$

$$Y = \begin{cases} +1 & \text{if } X \geq \theta \\ -1 & \text{if } X < \theta \end{cases}$$

เมื่อ X คือ ผลรวมของค่าน้ำหนักอินพุตคูณกับค่าอินพุตในเซลล์ประสาท

x_i คือ ค่าอินพุตของ i

w_i คือ น้ำหนักของอินพุต i,

n คือ ค่าตัวเลขจำนวนอินพุตของเซลล์ประสาท

Y คือ เอาต์พุตของเซลล์ประสาท

$$Y^{sigmoid} = \frac{1}{1+e^{-X}}$$

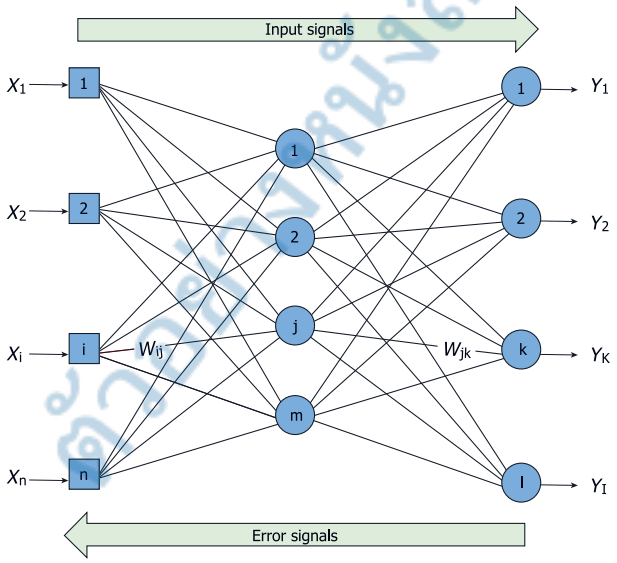
ฟังก์ชัน Sigmoid เป็นฟังก์ชันย่อยที่สามารถคำนวณได้ง่าย และรับรองได้ว่าค่าเอาต์พุตเครือข่ายเซลล์ประสาทจะมีค่าอยู่ระหว่าง 0 หรือ 1 เสมอ

กฎในการเรียนรู้ที่ใช้ในเครือข่ายเซลล์ประสาทเทียม Back-Propagation

กฎในการเรียนรู้ Back-Propagation จะประกอบด้วยเครือข่ายอย่างน้อยสามชั้น ดังแสดงในรูปที่ 16-8 โดยที่ i, j และ k เป็นการอ้างอิงเซลล์ประสาทในชั้นอินพุต, ชั้นที่ซ้อน และชั้นเอาต์พุต ตามลำดับ

สัญญาณอินพุต (Input Signal) $x_1, x_2, ..., x_n$ จะถูกส่งเข้าไปในเครือข่ายจากซ้ายไปขวา และสัญญาณความผิดพลาด (Error Signal) $e_1, e_2, ..., e_n$ จะส่งจากขวาไปซ้าย สัญลักษณ์ w_{ij} หมายถึง น้ำหนักในการเชื่อมต่อระหว่างเซลล์ประสาท i ในชั้นอินพุต และเซลล์ประสาท j ในชั้นที่ซ้อน และสัญลักษณ์ w_{jk} คือ น้ำหนักระหว่างเซลล์ประสาท j ในชั้นที่ซ่อน และเซลล์ประสาท k ในชั้นเอาต์พุต

รูปที่ 16-8
เครือข่ายเซลล์ประสาท Back-Propagation สามชั้น



การส่งค่าย้อนกลับ จะนำค่าความผิดพลาดที่เริ่มจากชั้นเอาต์พุตย้อนกลับไปยังชั้นที่ซ้อน ค่าความผิดพลาดเอาต์พุตจากเครือข่ายเซลล์ประสาท k ที่ลำดับการทำงาน p กำหนดได้ ดังนี้

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

โดยที่ $y_{d,k}(p)$ คือ ค่าเอาต์พุตของเซลล์ประสาท k ที่ลำดับการทำงาน p ที่ต้องการ

เซลล์ประสาท k คือ ส่วนของชั้นเอาต์พุตที่ต้องการ ดังนั้น สามารถใช้กระบวนการแบบตรงๆ ในการปรับน้ำหนัก w_{jk} โดยข้อเท็จจริงแล้ว กฎในการปรับน้ำหนักที่ชั้นเอาต์พุตจะเหมือนกับกฎในการเรียนรู้ของ Perceptron คือสมการที่ 16.2 ดังนี้