บทที่ 3

ปัญหาด้านเรขาคณิตเชิงคำนวณในงานวิจัย

ในบทนี้จะกล่าวถึงคำศัพท์ที่ใช้ภายในบท เพื่อให้เข้าใจความหมายของคำศัพท์ที่ตรงกัน และกล่าวถึงปัญหาทางด้านเรขาคณิตเชิงคำนวณในสองมิติที่นำมาใช้ในงานวิจัย โดยอธิบายลักษณะปัญหา และอัลกอริทึมที่ใช้ในการแก้ปัญหา และประสิทธิภาพของอัลกอริทึมแต่ละแบบ

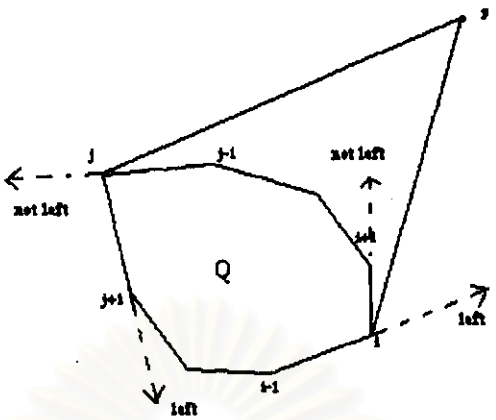
3.1 คำศัพท์และการดำเนินการที่ใช้ในงานวิจัย[1]

เนื่องจากเรขาคณิตเชิงคำนวณ เป็นการแก้ปัญหาเกี่ยวกับวัตถุเรขาคณิต ซึ่งอาจมีคำศัพท์บางคำซึ่งเป็นศัพท์เฉพาะ ดังนั้นเพื่อให้เกิดความเข้าใจในความหมายของคำศัพท์ที่ใช้ในการอธิบายขั้นตอนการแก้ปัญหาของอัลกอริทึม จึงแสดงคำศัพท์และความหมายของแต่ละคำดังนี้

1. *จุด (point)* แทนด้วยคู่ลำดับ (x,y) ค่า x และ y เป็นจำนวนเต็มใด ๆ
2. *เส้นตรง (line)* แทนด้วยจุดสองจุด p_1 และ p_2 ซึ่งสามารถเขียนในรูปผลรวมเชิงเส้น $\alpha p_1+(1-\alpha)p_2$ ซึ่ง $0\leq\alpha\leq1$
3. *การเลี้ยวซ้าย (turn left)* หรือ *เลี้ยวขวา (turn right)* เป็นการทดสอบจุดสามจุดที่อยู่ติดกัน $p_1p_2p_3$ ว่าจุด p_3 อยู่ทางซ้ายหรือทางขวาของเส้นตรง p_1 p_2 การทดสอบทำได้โดยการตรวจสอบว่า $p_1p_2p_3$ ทำให้เกิดวงจรในทิศทวนเข็มนาฬิกาหรือตามเข็มนาฬิกา ถ้าทวนเข็มนาฬิกา จะเป็นการเลี้ยวซ้าย ถ้าตามเข็มเป็นเลี้ยวขวา
4. *รูปหลายเหลี่ยม (polygon)* ในสองมิติ จะแทนด้วยลำดับของจุด $p_0,p_1,p_2,\ldots,p_n$ ซึ่ง รูปหลายเหลี่ยมคือ เซตจำกัดของส่วนของเส้นตรงซึ่งมีการใช้จุดปลายร่วมกันเพียงสองด้านเท่านั้น และจุดเริ่มต้นและจุดสุดท้ายเป็นจุดเดียวกันเป็นรูปปิด ส่วนของเส้นตรงที่เชื่อมต่อระหว่างจุดในรูปหลายเหลี่ยมเรียกว่าเส้นขอบ (edge) และจุดจะเรียกว่า จุดยอด (vertex) รูปหลายเหลี่ยมจะมีเส้นขอบและจุดยอดเท่ากัน
5. *รูปหลายเหลี่ยมนูน (convex polygon)* คือ รูปหลายเหลี่ยมซึ่งส่วนของเส้นตรงที่ลากเชื่อมต่อจุดสองจุดใด ๆ ของรูปหลายเหลี่ยมอยู่ภายในรูปหลายเหลี่ยมนั้น
6. *เส้นค้ำจุน (supporting line)* หรือ *เส้นสัมผัส (tangent line)* ของรูปหลายเหลี่ยมนูน Q คือเส้นตรงซึ่งผ่านจุดยอดของ Q เพียงจุดเดียว ซึ่งจุดทั้งหมดภายในรูปหลายเหลี่ยมจะอยู่ด้านใดด้านหนึ่งของเส้นตรง เส้นค้ำจุนที่ลากเชื่อมต่อระหว่างจุด p กับ Q สามารถหาได้โดย ทดสอบเส้นตรงที่ลากเชื่อมจุด p กับจุดบน Q ว่าเป็นเลี้ยวซ้ายหรือเลี้ยวขวา โดยจุด p_i ซึ่ง $pp_{i-1}p_i$ และ p p_ip_{i+1} มีการกลับกันจากเลี้ยวซ้ายเป็นเลี้ยวขวา หรือเลี้ยวขวาเป็นเลี้ยวซ้าย จุด p_i นั้นคือจุดที่เป็นจุดปลายของเส้นค้ำจุนที่ลากเชื่อมต่อกับ p ลักษณะของเส้นค้ำจุนแสดงดังรูปที่ 3-1 และจะเรียกจุด p_i นี้ว่าจุดสัมผัส (tangent point)

---

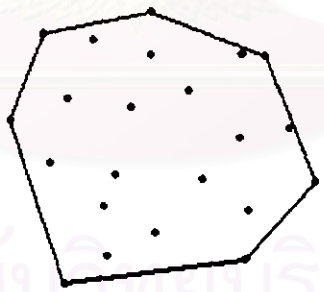
[1] Preparata and Shamos, Computational geometry : an introduction (New york: Springer-Verlag), p. 17.



รูปที่ 3-1 เส้นค้ำจุนของรูปหลายเหลี่ยม Q

3.2 *ปัญหาการหาเปลือกนูน (convex hull)*

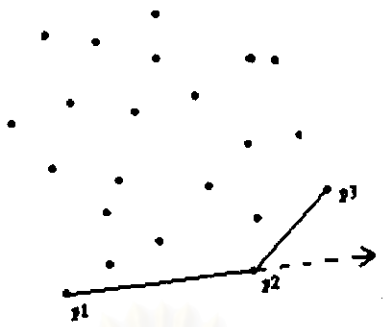
เปลือกนูนของเซตของจุด คือ รูปหลายเหลี่ยมนูนที่เล็กที่สุดที่ล้อมรอบจุดทั้งหมด ดังนั้นปัญหาการหาเปลือกนูนก็ คือการหาขอบเขตของเซตของเซตของจุดนั่นเอง ตัวอย่างภาพของเปลือกนูนแสดงดังรูปที่ 3-2 การหาเปลือกนูนมีความสำคัญในการประยุกต์ และยังเป็นเครื่องมือสำคัญในการแก้ปัญหาแบบอื่น ๆ ในด้านเรขาคณิตเชิงคำนวณด้วย ในส่วนนี้จะกล่าวถึงแนวคิดและลำดับขั้นตอนในการแก้ปัญหาด้วยอัลกอริทึมต่าง ๆ ซึ่งลำดับของจุดที่เก็บจะมีผลการการดำเนินการทดสอบบางอย่างของอัลกอริทึม ดังนั้นในทุกอัลกอริทึมจะเรียงลำดับจุดในทิศทวนเข็มนาฬิกา



รูปที่ 3-2 ภาพของเปลือกนูน

3.2.1 อัลกอริทึมแบบห่อของขวัญของ Javis (Javis's March)

แนวคิดของอัลกอริทึมเกิดจากการกำหนดเซตของจุด แล้วทำการตรวจสอบจุดทีละคู่ว่าเป็นขอบของเปลือกนูนหรือไม่ ซึ่ง Javis สังเกตว่าในกรณีที่หาขอบของเปลือกนูน pq ได้แล้วขอบต่อไปของเปลือกนูนจะต้องมีจุด p หรือ จุด q เป็นจุดปลาย ดังนั้นในขั้นแรกจะหาจุดที่อยู่บนเปลือกนูนก่อน โดยเริ่มจากจุดที่มีค่าแกน y ต่ำสุด ให้เป็น p_1 หลังจากนั้นจุดต่อไปของเปลือกนูนหาได้จากการคำนวณหาจุด p_2 ที่เส้นตรง p_1p_2 ทำมุมกับแกน x น้อยที่สุด แล้วหาจุด p_3 ซึ่งเส้นตรง p_2p_3 ทำมุมกับเส้นตรง p_1p_2 น้อยที่สุด การเปรียบเทียบมุมแสดงดังรูปที่ 3-3 แล้วทำการหาจุดต่อไปในลักษณะเดิม จนกระทั่งจุดที่เลือกเป็นจุดเริ่มต้นคือ p_1



รูปที่ 3-3 การเปรียบเทียบเพื่อหาค่ามุมที่น้อยที่สุด

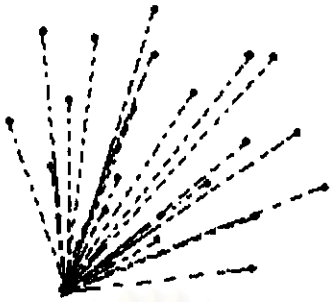
```
P แทน เซตของจุด  = { p_i }  , 0  ≤ i ≤ N

Angle(p1,p2,p3) เป็นฟังก์ชันที่คำนวณหาค่ามุมระหว่างเส้นตรง p1p2 และ เส้นตรง p2p3
PROCEDURE  Javis's March (P; N)
BEGIN
       AVisSync(N)                                                    (1)
       MinY ← FindMinY(P,N)
       Swap(1,MinY)
       p_N+1 ← i1      /*  Sentinel  */
       Min ← 1
       i ← 0
       REPEAT
         i ← i+1
         Min ← i
         FOR  j ← i+1 TO  N+1
           AVisSync(1)                                                (2)
           IF Angle(p_i-1,p_i,p_j) < Angle(p_i-1,p_i,p_Min) THEN  Min ← j
         NEXT  j
         Swap(i+1,Min)
       UNTIL  Min = N+1  /*  Meet first point  again  */
END
```

รูปที่ 3-4 อัลกอริทึมแบบห่อของขวัญของ Javis

อัลกอริทึมของ Javis ถ้ามีจำนวนจุดยอดของเปลือกนูน เป็น h และเวลาในการหาจุดยอดในแต่ละรอบเป็นเชิงเส้น ดังนั้น Javis จะใช้เวลา O(hN) และในกรณีที่แย่ที่สุด คือจุดทั้งหมดเป็นจุดยอดของเปลือกนูน จะใช้เวลา $O(N^2)$

3.2.2 อัลกอริทึมแบบกราดตรวจของ Graham (Graham's Scan)

ขั้นตอนในการแก้ปัญหาเริ่มต้นโดยเรียงลำดับจุดตามมุมที่ทำระหว่างแกน x กับเส้นตรงที่เชื่อมจุดนั้นกับจุดอ้างอิงจุดหนึ่ง ในที่นี้เลือกจุดที่มีค่า y ต่ำสุดเป็นจุดอ้างอิง ในกรณีที่จุดสองจุดทำมุมเท่ากันจะเรียงลำดับตามระยะทางที่ทำกับจุดอ้างอิง การเรียงลำดับแสดงดังรูปที่ 3-5 หลังจากการเรียงลำดับจะทำการกราดตรวจตามลำดับที่เรียงไว้ โดยในการพิจารณา จะพิจารณาจุดทีละสามจุดตามลำดับที่อยู่ติดกัน กำหนดให้เริ่มแรกมีจุดสามจุดอยู่บนเปลือกนูนคือ $p_1p_2p_3$ เมื่อเพิ่มจุด p_4 จะพิจารณาว่า ถ้า $p_1p_2p_3$ เป็นการเลี้ยวขวาจะกำจัดจุด p_2 ออกจากเปลือกนูน แล้วทดสอบ $p_1p_3p_4$ ในลักษณะเดียวกัน ถ้า $p_1p_3p_4$ เป็นการเลี้ยวซ้ายจะพิจารณาจุดต่อไป ในการทดสอบจะตรวจสอบทีละสามจุดตามลำดับจนครบทุกจุด



รูปที่ 3-5 การเรียงลำดับของจุดตามมุมเทียบกับจุดอ้างอิง

P แทน เซตของจุด = $\{ p_i \}$, $0 \le i \le N$

Angle(p) เป็นฟังก์ชันสำหรับคำนวณหาค่ามุมที่จุด p ทำกับแกน x

QSortAngle(P;l,u) เป็นกระบวนงานสำหรับเรียงลำดับจุดตามมุมเมื่อเทียบกับแกน x

TurnLeft(p_1,p_2,p_3) เป็นกระบวนงานตรวจสอบว่าการเรียงลำดับของจุดที่ $p_1p_2p_3$ เป็นเลี้ยวซ้ายหรือไม่

```
PROCEDURE  Graham 's scan(P;N)
BEGIN
       AVisSync(N)                                              (1)
       MinY ← FindMinY(P;N)
       Swap(1,MinY)
       QSortAngle(P;2,N)
       m ← 3
       FOR i ← 4 TO N
         WHILE NOT (TurnLeft(Pm-1,Pm,Pi))
            AVisSync(1)                                         (2)
            m ← m-1
         END WHILE
         m←m+1
         SWap(i,m)
         AVisSync(1)                                            (3)
       NEXT  i
END
PROCEDURE QSortAngle(P;l,u)
BEGIN
       mid ← (l+u) / 2
       REPEAT
         i ← l ;   j ← u
         WHILE (Angle(Pi) < Angle(Pmid))
           AVisSync(1)                                          (4)
           i ← i+1
         END WHILE
         WHILE (Angle(Pj) > Angle( Pmid))
           AVisSync(1)                                          (5)
           j ← j-1
         END WHILE
         IF i <= j THEN Swap(i,j)
         AVisSync(1)                                            (6)
       UNTIL  i > j
       IF l < j THEN  QSortAngle(P;l,j)
       IF i < u THEN  QSortAngle(P;i,u)
END
```

รูปที่ 3-6 อัลกอริทึมแบบกราดตรวจของ Graham

สำหรับอัลกอริทึมแบบกราดตรวจ เวลาของอัลกอริทึมที่ใช้ส่วนใหญ่จะขึ้นอยู่กับการเรียงลำดับ ซึ่งขั้นตอนการเรียงลำดับใช้เวลา O(n log n) และ ในขั้นตอนการกราดตรวจจะใช้เวลา O(n) ดังนั้น เวลาในการประมวลผลของอัลกอริทึมแบบกราดตรวจของ Graham คือ O(n log n)

3.2.3 อัลกอริทึมแบบค่อย ๆ เพิ่มจุด (Incremental)

การแก้ปัญหาเกิดจากแนวความคิดในการสร้างเปลือกนูนที่มี n จุด จากเปลือกนูนที่มี n-1 จุด ขั้นตอนการแก้ปัญหาจะเริ่มจากการสร้างเปลือกนูนจากจุดสามจุดใด ๆ แล้วเพิ่มจุดเข้าไปทีละจุดซึ่งมีกรณีที่พิจารณาสองกรณี คือ จุดที่เพิ่มเข้าไปใหม่อยู่ภายในเปลือกนูน หรืออยู่นอกเปลือกนูน ถ้าอยู่ในเปลือกนูนก็ไม่ต้องนำมาพิจารณาต่อ หากจุดนั้นอยู่นอกเปลือกนูนต้องขยายเปลือกนูนออก โดยการหาเส้นตรงที่เป็นเส้นค้ำจุนจากจุดนั้นกับเปลือกนูน จากนั้นลบจุดยอดของเปลือกนูนที่อยู่ระหว่างเส้นค้ำจุดทั้งสองเส้นที่หาได้ แล้วแทรกจุดที่ต้องการเพิ่มเข้าไปแทน การขยายเปลือกนูนแสดงดังรูปที่ 3-7 ในการขยายเปลือกนูนจะลบจุด p_3 และ p_4 แล้วแทรก p_6 เข้าไปในระหว่าง p_2 และ p_5

รูปที่ 3-7 การขยายเปลือกนูนในกรณีที่จุดที่เพิ่มเข้าไปใหม่ไม่อยู่ในเปลือกนูน

```
P แทน เซตของจุด =  { p1 }  , 0 ≤ i ≤ N
InConvex(H;p) เป็นกระบวนงานตรวจสอบว่า จุด p อยู่ภายในรูปหลายเหลี่ยนนูน H หรือไม่
FindInsertPoint(H;a,b,p) เป็นกระบวนงานในการหาตำแหน่งแทรกของจุด p โดยคืนค่าตำแหน่งแทรก
ระหว่างจุด a และ b
InsertPoint(H;a,b,p) เป็นกระบวนงานสำหรับการลบจุดที่อยู่ระหว่าง a และ b แล้วแทรกจุด p ใน
ระหว่างจุด a และ b
PROCEDURE Incremental(P;N)
BEGIN
      H = { p1, p2 , p3 }     /* build convex hull from first 3 points  */
      FOR i ← 4   TO   N
        AVisSync(1)                                          (1)
        IF NOT InConvex(H;pi) THEN
          FindInsertPoint(H;a,b,pi)
          InsertPoint(H;a,b,pi)
        END IF
      NEXT i
END
```

รูปที่ 3-8 อัลกอริทึมแบบค่อย ๆ เพิ่มจุด

```
FUNCTION InConvex(H;p)
BEGIN
      FOR i ← 1   TO   Count(H)
        AVisSync(1)                                              (2)
        IF  NOT TurnLeft(pi,pi+1,p)  THEN   RETURN (false)
      NEXT i
      RETURN (true)
END
PROCEDURE  FindInsertPoint(H;a,b,p)
BEGIN
      FOR i ←1   TO   Count(H)
        AVisSync(1)                                              (3)
        IF XOR(TurnLeft(pi-1,pi,p),TurnLeft(pi,pi+1,p)) THEN
          a ← i
          EXIT FOR
        END IF
      NEXT i
      FOR j ← i   TO   Count(H)
        AVisSync(1)                                              (4)
        IF XOR(TurnLeft(pj-1,pj,p),TurnLeft(pj,pj+1,p)) THEN
          b ← j
          EXIT FOR
        END IF
      NEXT j
END
```

รูปที่ 3-8 อัลกอริทึมแบบค่อย ๆ เพิ่มจุด (ต่อ)

อัลกอริทึมแบบค่อย ๆ เพิ่มจุด จะใช้เวลาสำหรับแต่ละรอบ O(k) ในกรณีที่มี k จุดบนเปลือกนูน ให้ T(n) คือ เวลาในการคำนวณหาเปลือนนูนจากจุด n จุด ในกรณีที่ไม่มีจุดใดอยู่ในเปลือกนูนเลย จะใช้เวลา $T(n) = T(n-1) + O(n) = O(n^2)$

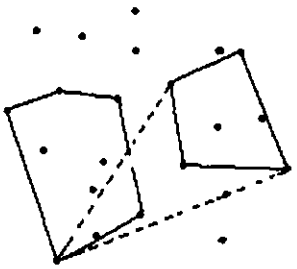
3.2.4 อัลกอริทึมแบบแบ่งแยกแล้วเอาชนะ (Divide-and-conquer)

การแก้ปัญหาจะเริ่มต้นโดยการแบ่งเซตของข้อมูลเข้าออกเป็นสองเซตย่อยเท่า ๆ กัน แล้วคำนวณหาเปลือกนูนของเซตย่อยในลักษณะการเรียกซ้ำ จากนั้นนำผลลัพธ์ที่ได้จากเซตย่อยมาผสานกัน ขั้นตอนการผสานเปลือกนูนจะเลือกจุดที่มีค่าแกน y ต่ำสุดหนึ่งจุดจากเปลือกนูนทั้งสอง แล้วหาเส้นค้ำจุนจากจุดนั้นกับเปลือกนูนที่ไม่มีจุดนี้อยู่ แล้วลบจุดที่อยู่ระหว่างเส้นค้ำจุนทั้งสอง การหาเส้นค้ำจุนสำหรับการผสานเปลือกนูนแสดงดังรูปที่ 3-9 ในขั้นตอนนี้จะได้รายการของจุดที่มีการเรียงลำดับตามมุมเมื่อเทียบกับจุดที่เลือกขึ้นมา กการที่ได้รายการสองรายการที่มีการเรียงลำดับตามมุมทำให้สามารถผสานสองรายการที่ได้แล้วใช้วิธีแบบกราดตรวจของ Graham ในการหาเปลือกนูน การผสานนี้เป็นวิธีของ Preparata และ Shamos[2] นอกจากนี้ยังมีวิธีของ Preparata และ Hong[3] ซึ่งมีความแตกต่างในเรื่องของวิธีการผสานเปลือกนูน ผู้ที่สนใจสามารถศึกษาเพิ่มเติมได้

---

[2] Preparata and Shamos, Computational geometry : an introduction ,p. 116.

[3] Preparata and Hong, "Convex hull of finite sets of points in two and three dimensions," Communication of the ACM 20(1977) :87-93



รูปที่ 3-9 เส้นค้ำจุนสำหรับการผสานเปลือกนูน

P แทน เซตของจุด = $\{ p_i \}$, $0 \le i \le N$

l และ u แทน ขอบล่างและขอบบนของข้อมูลตามลำดับ

k_0 แทนจำนวนข้อมูลที่น้อยที่สุดในการสร้างเปลือกนูน

FindInsertPoint(H;a,b,p) เป็นกระบวนงานหาตำแหน่งแทรกของจุด p โดยคืนค่าตำแหน่งแทรกระหว่างจุด a และ b

Scan of Graham(H) เป็นกระบวนงานเหมือนการกราดตรวจของ Graham

```
FUNCTION Divide (P;l,u)
BEGIN
       IF  Count(P) <= k0  THEN
         Form Hull (H) with k0 points
         RETURN (H)
       END IF
       mid ← (l+u) / 2
       H1 ← Divide(P;l,mid)
       H2 ← Divide(P;mid,u)
       MergeHull(H1,H2)
END
PROCEDURE MergeHull(H1,H2)
BEGIN
       pMin← Minimum Y value point of Hi     , i = 1,2
       HInsert ← H1 with minimum Y value ;  HCut ← another H1
       FindInsertPoint(HCut;a1,b, pMin)
       DeletePoint(HCut,a,b)      /* delete point in HCut between a and b  */
       H ← Merge2Lst(HCut,HInsert)
       Scan of Graham(H)
END
PROCEDURE  FindInsertPoint(H;a,b,p)
BEGIN
       FOR  i ←1   TO   Count(H)
         AVisSync(1)                                              (1)
         IF XOR(TurnLeft(pi-1,pi,p),TurnLeft(pi,pi+1,p)) THEN
           a ← i
           EXIT FOR
         END IF
       NEXT
       FOR  j ← i   TO   Count(H)
         AVisSync(1)                                              (2)
         IF XOR(TurnLeft(pj-1,pj,p),TurnLeft(pj,pj+1,p)) THEN
           b ← j
           EXIT FOR
         END IF
       NEXT j
END
```

รูปที่ 3-10 อัลกอริทึมแบบแบ่งแยกแล้วเอาชนะ

```
FUNCTION Merge2Lst(H1,H2)
BEGIN
       k ← 0
       WHILE (i <= Count(H1) AND (j <= Count(H2)
         AVisSync(1)                                          (3)
         IF Angle(H1i) < Angle(H2j)    THEN
           Hok ← H1i
           i ← i+1; k ← k+1
         ELSE
           Hok ← H2j
           j← j+1; k ← k+1
         ENDIF
       END WHILE
       IF (i  > Count(H1)  THEN
         WHILE (j < Count(H2))
             AVisSync(1)                                      (4)
             Hok ←  H2j;  j← j+1; k ← k+1
         END WHILE
       ELSE
         WHILE (i < Count(H1))
           AVisSync(1)                                        (5)
           Hok ←  H1i
           i← i+1; k ← k+1
         END WHILE
       END IF
       RETURN (H)
END
```

รูปที่ 3-10 อัลกอริทึมแบบแบ่งแยกแล้วเอาชนะ (ต่อ)

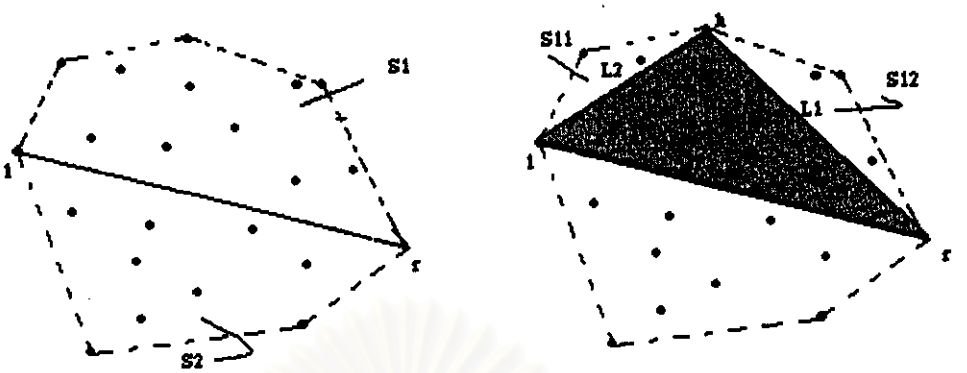
ให้ T(n) เป็นเวลาที่ใช้ในการหาเปลือกนูนจาก n จุด ขั้นตอนการผสานเปลือกนูนใช้เวลา O(n) และขั้นตอนในการหาเปลือกนูนของ n จุด เกิดจากการหาเปลือกนูนของ n/2 จุด 2 ชุดแล้วนำมาผสานกัน ดังนั้น $T(n) = 2\,T(n/2) + O(n) = O(n \log n)$

3.2.5 การหาเปลือกนูนแบบเร็ว (Quick Hull)[4]

การเริ่มต้น จะแบ่งเซตของจุดออกเป็นเซตย่อยสองเซต การแบ่งเริ่มต้นจะกำหนดโดยเส้นตรงซึ่งผ่านจุด l และ r ซึ่งเป็นจุดที่มีค่าแกน x น้อยที่สุดและมากที่สุดตามลำดับ ให้ S_1 เป็นเซตย่อยของจุดซึ่งอยู่บนหรือเหนือเส้น l และ r S_2 เป็นเซตย่อยของจุดที่อยู่ใต้เส้น หลังจากนั้นจะทำขั้นตอนต่อไปในแต่ละเซต S_1 และ S_2 โดยจะหาจุด h ในเซต S_1 ซึ่งสามเหลี่ยม hlr เป็นสามเหลี่ยมที่มีพื้นที่มากที่สุด ซึ่งนั่นคือ มุม hlr มีค่ามากที่สุด ดังนั้นจุด h เป็นจุดที่อยู่บนเปลือกนูน ซึ่งถ้าหากลากเส้นที่ขนานกับเส้นตรง lr และผ่านจุด h ไม่มีจุดใน S ที่อยู่เหนือเส้นตรงนี้ หลังจากนั้นจึงสร้างเส้นตรงสองเส้น เป็นเส้นตรง L_1 ซึ่งเป็นเส้นตรงซึ่งลากจากจุด r ไปหาจุด h และเส้นตรง L_2 ลากจาก h ไป l สำหรับแต่ละจุดใน S_1 จะไม่มีจุดไหนที่อยู่ทางด้านขวาของทั้ง L_1 และ L_2 ขณะที่จุดที่อยู่ด้านซ้ายของทั้งสองเส้นจะเป็นจุดที่อยู่ภายในสามเหลี่ยม lrh และสามารถตัดออกจากการพิจารณาได้ และสร้างเซตของจุด S_{1_1} เป็นจุดซึ่งอยู่ทางขวาของเส้น L_1 แต่อยู่ทางซ้ายของ L_2 และสร้างเซตของจุด S_{1_2} จากจุดซึ่งอยู่ทางขวาของ L_2 และพิจารณาเซตของจุดที่ได้ในลักษณะการเรียกซ้ำ การแบ่งเซตของจุดแสดงดังรูปที่ 3-11 หลังจากที่ได้ผลลัพธ์ของเซต S_1 แล้วพิจารณาในลักษณะเดียวกับเซต S_2

---

[4] Ibid. p. 112.



รูปที่ 3-11 การแบ่งเซตของจุดในการหาเปลือกนูนแบบเร็ว

```
กำหนด P แทน เซตของจุด = { pi } , 0 ≤ i ≤ N
l และ u แทนขอบเขตล่างและขอบบนของข้อมูลตามลำดับ
Distance(p;l,r2) เป็นฟังก์ชันคำนวณระยะห่างระหว่างจุด p กับเส้นตรง lr
RightPoint(P;l,r) เป็นฟังก์ชันหาเซตของจุดที่อยู่ทางด้านขวาของเส้นตรง lr
MaxDistance(P;l,r) เป็นฟังก์ชันที่หาจุดที่มีค่าระยะห่างที่มากที่สุดกับเส้นตรง lr
FUNCTION QuickHull(P;l,r)
BEGIN
      IF  P = {l,r} THEN  RETURN (l,r)
      h ← MaxDistance(P;l,r)
      P1 ← RightPoint(P;r,h)
      P2 ← RightPoint(P;h,l)
      RETURN Concat(QuickHull(P1;r,h),QuickHull(P2;h,l)
END
FUNCTION MaxDistance(P;l,r)
BEGIN
      maxdist ← 1
      FOR i ← 2   TO   Count(P)
        AVisSync(1)                                              (1)
        IF Distance(pi,l,r) > Distance(pmaxdist,l,r)  THEN   maxdist ← i
      NEXT i
END
FUNCTION RightPoint(P;l,r)
BEGIN
      FOR  i  ← 1   TO   Count(P)
        AVisSync(1)                                              (2)
        IF TurnLeft(l,r,pi)   THEN  exclude pi from P
      NEXT i
      RETURN (P)
END
```

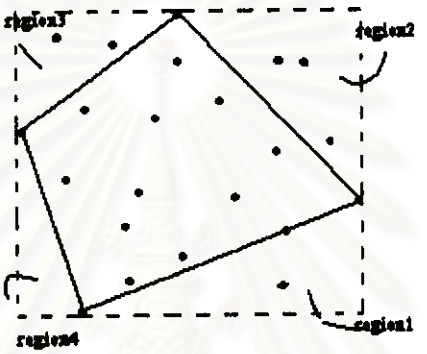
รูปที่ 3-12 อัลกอริทึมการหาเปลือกนูนแบบเร็ว

ในการหาเปลือกนูนแบบเร็วใช้เวลาในการหาเส้นตรง lr คือ O(n) และใช้เวลา n ในการหาจุด h เวลาที่ใช้สำหรับการเรียกซ้ำจะขึ้นอยู่กับจำนวนข้อมูลของ S_1 และ S_2 ให้เป็น α และ β ดังนั้นใช้เวลาในการหาเปลือกนูน n จุด $T(n) = O(n) + T(\alpha)+T(\beta)$ ในกรณีที่เซต S_1 และ S_2 ได้รับการแบ่งเท่า ๆ กัน คือ n/2 ดังนั้น

$T(n) = 2\,T(n/2) + O(n) = O(n \log n)$

3.2.6 อัลกอริทึมแบบกำจัด[5]

ขั้นตอนในการหาเปลือกนูน คือ หาจุดสุดขีดสี่จุด คือ จุดที่มีค่า x มากที่สุดและน้อยที่สุด และจุดที่มีค่า y มากที่สุดและน้อยที่สุดจากเซตของจุดที่กำหนดให้ แล้วตัดจุดทั้งหมดที่อยู่ภายในสี่เหลี่ยมที่เกิดจากจุดสี่จุดนี้ออกจากการพิจารณาซึ่งการตัดจุดนี้เป็นการใช้หลักการการกำจัด จากนั้นจะแบ่งจุดที่เหลืออยู่ออกเป็นสี่กลุ่มตามจุดสุดขีดที่หาได้ดังรูปที่ 3-13 แล้วเรียงลำดับจุดในแต่ละบริเวณตามค่าแกน x ดังนี้ ในบริเวณที่หนึ่งและสี่เรียงลำดับจากน้อยไปหามาก บริเวณที่สองและสามเรียงลำดับจากมากไปหาน้อย แล้วหาเส้นขอบของเปลือกนูนโดยใช้การพิจารณาในมีลักษณะเดียวกันกับการพิจารณาจุดของ Graham



รูปที่ 3-13 การแบ่งบริเวณในการหาเปลือกนูน

P แทน เซตของจุด = { p_i } , 0 ≤ i ≤ N

FindExtreamPoint(P;N) เป็นฟังก์ชันในการคำนวณหาค่าจุดสุดขีดทั้งสี่จุดโดยคืนค่าที่แถวลำดับ E

RightPoint(P;p_1,p_2) เป็นฟังก์ชันหาเซตของจุดที่อยู่ทางด้านขวาของเส้นตรง p_1p_2

QSortX (P;l,u,Order) เป็นกระบวนงานเรียงลำดับจุดในเซต P ตามแกน x ตาม Order ที่ระบุ

Scan of Graham(H) เป็นกระบวนงานเหมือนการกราดตรวจของ Graham

```
PROCEDURE FastHull(P;N)
BEGIN
       AVisSync(2*N)                                          (1)
       E ← FindExtreamPoint(P;N)
       FOR i ← 1   TO   4
         P_i ← RightPoint(P;E_i,E_i+1)
         SortInRegion(P_i;Order)
         Scan of Graham(P_i)
       NEXT i
END
FUNCTION RightPoint(P;l,r)
BEGIN
       FOR  i  ← 1   TO   Count(P)
         AVisSync(1)                                          (2)
         IF TurnLeft(l,r,p_i)   THEN  exclude p_i from P
       NEXT i
       RETURN (P)
END
```

รูปที่ 3-14 อัลกอริทึมการหาเปลือกนูนแบบกำจัด

[5] AKL and Toussaint, "A fast convex hull algorithm," Inform. Process. Lett. 7(1978), p.219-222.

```
PROCEDURE QSortX (P,l,u,Order)
BEGIN
      mid ← (l+u) / 2
      REPTEAT
        i ←l ;       j ← u
        WHILE  Cmp(pi.x, pmid.x,Order) < 0
         AVisSync(1)                                    (3)
         i ← i+1
        END WHILE
        WHILE Cmp(pj.x, pmid.x,Order) > 0
          AVisSync(1)                                   (4)
          j ← j-1
        END WHILE
        IF i <= j   THEN   Swap(i,j)
        AVisSync(1)                                     (5)
      UNTIL i> j
      IF l < j    THEN   QSortX (P;l,j,Order)
      IF i < u    THEN   QSortX (P;i,u,Order)
END
```

รูปที่ 3-14 อัลกอริทึมการหาเปลือกนูนแบบกำจัด (ต่อ)

สำหรับเวลาที่ใช้ส่วนใหญ่เป็นการเรียงลำดับ ซึ่งขั้นตอนการเรียงลำดับใช้เวลา O(n log n) สำหรับขั้นตอนอื่น ๆ ใช้ O(n) ดังนั้นเวลาในการประมวลผลของอัลกอริทึมเป็น O(n log n)

3.3 *ปัญหาการค้นหาในพิสัย (Range Searching)*

การค้นหาในพิสัย เป็นหนึ่งในปัญหาเกี่ยวกับการค้นหาเชิงเรขาคณิต ซึ่งการค้นหาในพิสัยในที่นี้เป็นการสอบถามข้อมูลที่เป็นจุด โดยมีพิสัยของการสอบถามเป็นรูปสี่เหลี่ยม และลักษณะของปัญหาเป็นปัญหาการนับจำนวนจุดในช่วงที่กำหนด (range counting) การแก้ปัญหาส่วนใหญ่จะประกอบด้วยสองขั้นตอนคือ การประมวลผลก่อน คือการสร้างข้อมูลให้อยู่ในโครงสร้างที่สนับสนุนการค้นหา และขั้นตอนการค้นหาซึ่งเป็นการคืนจำนวนจุดที่อยู่ภายในพิสัยที่กำหนด ยกเว้นการค้นหาโดยวิธีการค้นหาแบบลำดับ ซึ่งไม่มีขั้นตอนของการประมวลผลก่อน

3.3.1 การค้นหาแบบลำดับ

วิธีนี้ไม่มีขั้นตอนการประมวลผลก่อน ในการค้นหาจะทำการตรวจสอบแต่ละจุดตามลำดับว่าอยู่ภายในพิสัยที่กำหนดหรือไม่

```
P แทน เซตของจุด = { pi }  , 0 ≤ i ≤ N

InsideRect(p,Rect) เป็นฟังก์ชันที่ตรวจสอบว่าจุด p อยู่ภายในสี่เหลี่ยม Rect หรือไม่
PROCEDURE Search(P;N,Rect)
BEGIN
      FOR i ← 1 TO N
        AVisSync(1)                                     (1)
        IF (InsideRect(pi,RECT) THEN  Output (pi)
      NEXT i
END
```

รูปที่ 3-15 อัลกอริทึมการค้นหาแบบลำดับ

3.3.2 วิธีกริด (Grid Method)[6]

งานในส่วนประมวลผลก่อนจะเป็นการแบ่งระนาบออกเป็นสี่เหลี่ยมเล็ก ๆ เรียกว่ากริดแล้วเก็บรายการจุดที่อยู่ในสี่เหลี่ยมเดียวกันไว้ในรายการเดียวกัน ในส่วนนี้จะมีที่เก็บข้อมูลซึ่งทำหน้าที่เหมือนสารบบสำหรับชี้ตำแหน่งของรายการของจุดของแต่ละกริด และที่เก็บรายการของจุด สิ่งที่ต้องพิจารณา คือการหาของขนาดของกริดที่เหมาะสม หากสี่เหลี่ยมที่เป็นกริดมีขนาดใหญ่จะทำให้ได้รายการของจุดที่ยาวเกินไป หรือถ้ากริดมีขนาดเล็กจะมีกริดจำนวนมากที่ไม่มีจุดอยู่ โดยทั่วไปเลือกขนาดของกริดเป็นค่าคงที่ที่เป็นอัตราส่วนกับจำนวนจุดดังนี้

จำนวนจุด N จุด ต้องการให้มี M จุดต่อ 1 หนึ่งกริด ดังนั้นจะมีการแบ่งประมาณ N/M ชุด

จำนวนกริด คือ (ค่าแกนสูงสุด / ขนาดกริด)x(ค่าแกนสูงสุด / ขนาดกริด)

(ค่าแกนสูงสุด / ขนาดกริด)x(ค่าแกนสูงสุด / ขนาดกริด) = N/M

ขนาดกริด x ขนาดกริด = (ค่าแกนสูงสุด x ค่าแกนสูงสุด) x M/N

ขั้นตอนการค้นหาจะคำนวณหากริดที่ตัดกับสี่เหลี่ยมที่ต้องการค้นหา แล้วอ่านตำแหน่งเริ่มต้นของรายการจากสารบบ แล้วทำการค้นหารายการของจุดในละกริดแบบลำดับ

```
p แทน เซตของจุด = { pi } , 0 ≤ i ≤ N
GridTable (gSize,gSize) เป็นตารางที่แทนสารบบของรายการในกริด
InsideRect(p,Rect) เป็นฟังก์ชันที่ตรวจสอบว่าจุด p อยู่ภายในสี่เหลี่ยม Rect หรือไม่
PROCEDURE Grid Preprocess(P;N)
BEGIN
       FOR i ← 1 TO N
         gX ← pi.x / gSize
         gY ← pi.y / gSize
         pNode↑.dat ← pi
         pNode↑.next ← GridTable(gX,gY)
         GridTable(gX,gY) ←pNode
         AVisSync(1)                                          (1)
       NEXT i
END
PROCEDURE  Grid Search(GridTab;Rect)
BEGIN
       StartX ←  Rect.x1 / gSize;        EndX ←  Rect.x2 / gSize
       StartY ←  Rect.y1 / gSize;        EndY ←  Rect.y2 / gSize
       FOR i ←  StartX TO EndX
         FOR j ←  StartY TO EndY
           pNode ←  GridTable(i,j)
           WHILE pNode <> NIL
             AVisSync(1)                                      (2)
             IF InsideRect( pNode ↑.dat , Rect)  THEN  Output(pNode↑.dat)
             pNode ←  pNode↑.next
           END WHILE
         NEXT j
       NEXT i
END
```

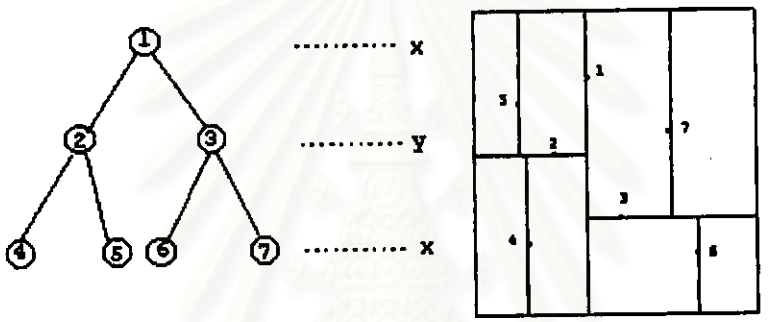
รูปที่ 3-16 อัลกอริทึมการใส่จุดลงในตารางและการค้นหา

---

[6] Sedgewick, Algorithms (MA : Addison-Wesley, 1988), p. 377-379.

3.3.3 ต้นไม้แบบสองมิติ (2 Dimensional Tree)[7]

ต้นไม้สองมิติ คือโครงสร้างข้อมูลที่มีลักษณะคล้ายต้นไม้แบบทวิภาค (binary tree) โดยในแต่ละโหนด (node) จะแทนจุดแต่ละจุดในระนาบ และมีคีย์เป็นค่าแกน x หรือแกน y ของจุด แนวคิด คือการแบ่งระนาบด้วยเส้นตรงที่ขนานกับแกน x และ แกน y ตัวอย่างเช่นถ้าโหนดราก (root) ใช้ค่าแกน x เป็นคีย์ จุดที่อยู่ทางซ้ายของรากจะอยู่ในต้นไม้ย่อยทางซ้าย และจุดทั้งหมดที่อยู่ทางขวาของรากจะอยู่ในต้นไม้ย่อยทางขวา ซึ่งเป็นการแบ่งระนาบสำหรับการค้นหาออกเป็นสองส่วน ตัวอย่างของต้นไม้สองมิติ แกนที่ใช้เป็นคีย์ และการแบ่งระนาบ แสดงดังรูปที่ 3-17



รูปที่ 3-17 ตัวอย่างของต้นไม้สองมิติ

งานในส่วนของการประมวลผลก่อน เริ่มต้นโดยเลือกค่าแกนที่จะใช้เป็นค่าคีย์ก่อน ในที่นี้จะใช้ค่าแกน x เป็นคีย์ของราก และต้นไม้ในระดับต่อไปจะใช้ค่าแกน y เป็นค่าคีย์ และสลับค่าคีย์ที่ใช้เป็นแกน x และ y สลับกันไปในต้นไม้แต่ละระดับ ดังนั้นหลังจากโหนดรากสำหรับจุดต่อไปที่แทรกเข้าไปในต้นไม้ ถ้ามีค่าแกน x น้อยกว่าจะลงไปทางซ้าย และถ้ามากกว่าจะลงไปทางขวา แล้วจุดที่แทรกใหม่จะใช้ค่าแกน x เป็นค่าคีย์ และทำในลักษณะเดียวกันสำหรับการแทรกจุดที่เหลือโดยการแทรกจะเปรียบเทียบค่าคีย์ที่เป็นแกน x และแกน y สลับกันไป จนกระทั่งพบโหนดที่เป็นใบ

ในการค้นหาจะเริ่มต้นเปรียบเทียบตั้งแต่โหนดราก โดยทดสอบว่าโหนดนั้นอยู่ภายในพิสัยหรือไม่ และตรวจสอบว่าขอบเขตของโหนดในระดับล่างมีการตัดกันกับพิสัยที่ต้องการค้นหาหรือไม่ ถ้ามีการตัดกันจะลงไปค้นหาในโหนดระดับล่างนั้น

---

[7] Bentley, "Multidimensional binary search trees used for associative searching," Communications of the ACM 18(1975), p.509-516.

p แทน เซตของจุด = { p_i } , $0 \le i \le N$

InsideRect(p,Rect) เป็นฟังก์ชันที่ตรวจสอบว่าจุด p อยู่ภายในสี่เหลี่ยม Rect หรือไม่

Successor(p,q) เป็นฟังก์ชันทดสอบว่าค่าคีย์ของโหนด p มากหรือน้อยกว่าโหนด q

BoundIntersectRegion(B,Rect) เป็นฟังก์ชันที่ทดสอบว่าขอบเขต B กับสี่เหลี่ยม Rect มีบริเวณร่วมกันอยู่หรือไม่

Adjust(BH,BL,p) เป็นกระบวนงานสำหรับปรับค่าขอบเขตตามค่าคีย์ของโหนด p ที่ค้นหาอยู่

```
PROCEDURE BuildTree(pNode)
BEGIN
       IF pRoot = NIL THEN
         pRoot ← pNode
         EXIT SUB
       END IF
       son ←  pRoot
       REPEAT
         q ←  son
         succ  ←Successor(pNode,q)                                    (1)
         AVisSync(1)
         IF succ = -1   THEN        /* insert key less than current node */
           son  ←  q↑.left
         ELSE IF succ = 1   THEN  /* insert key greater than current node */
           son ← q↑.right
       END IF
       UNTIL son  = NIL
       IF succ = -1 THEN
         q↑.left←pNode
       ELSE
         q↑.right ← pNode
       END IF
       IF q↑.discrim = coordX THEN       /* alternate key in each level*/
         pNode↑.discrim ← coordY
       ELSE
         pNode↑.discrim ← coordX
       END IF
END
PROCEDURE 2DSearch(p,Rect,B)
BEGIN
       AVisSync(1)                                                    (2)
       IF InsideRect(p↑.dat,rect) THEN   Output(p)
       Adjust (BH,BL,p↑.key)      /* Change bounding block  */
       AVisSync(1)
       IF p↑.left <> NIL AND BoundIntersectRegion(BL,Rect) THEN
         2DSearch(p↑.left,Rect,BL)
       IF p↑.right <> NIL AND BoundIntersectRegion(BH,rect) THEN
         2DSearch(p↑.right,Rect,BH)
END
```

รูปที่ 3-18 อัลกอริทึมการสร้างต้นไม้สองมิติและการค้นหา

3.3.4 ต้นไม้ที่แกนที่ใช้เป็นคีย์เป็นตัวเลขสุ่ม

วิธีการในการสร้างต้นไม้เหมือนต้นไม้สองมิติ แต่การเลือกแกนที่ใช้เป็นค่าคีย์สำหรับโหนดที่เพิ่มเข้าไปใหม่จะใช้ตัวเลขสุ่ม ดังนั้นค่าแกนที่ใช้เป็นคีย์ในแต่ละระดับอาจเป็นแกน x หรือแกน y ก็ได้ ส่วนการค้นหาจะเหมือนกับในต้นไม้สองมิติ

P แทน เซตของจุด = { p_i } , $0 \le i \le N$

InsideRect(p,Rect) เป็นฟังก์ชันที่ตรวจสอบว่าจุด p อยู่ภายในสี่เหลี่ยม Rect หรือไม่

Successor(p,q) เป็นฟังก์ชันทดสอบว่าค่าคีย์ของโหนด p มากหรือน้อยกว่าโหนด q

BoundIntersectRegion(B,Rect) เป็นฟังก์ชันที่ทดสอบว่าขอบเขต B กับสี่เหลี่ยม Rect มีบริเวณร่วมกันอยู่หรือไม่

Adjust(BH,BL,p) เป็นกระบวนงานสำหรับปรับค่าขอบเขตตามค่าคีย์ของโหนด p ที่ค้นหาอยู่

```
PROCEDURE BuildTree(pNode)
BEGIN
       IF pRoot = NIL THEN
         pRoot ← pNode
         EXIT SUB
       END IF
       son ←  pRoot
       REPEAT
         q ←  son
         succ  ←Successor(pNode,q)
         AVisSync(1)                                              (1)
         IF succ = -1   THEN       /* insert key less than current node */
           son  ←  q↑.left
         ELSE IF succ = 1   THEN  /* insert key greater than current node */
           son ← q↑.right
       END IF
       UNTIL son  = NIL
       IF succ = -1 THEN
         q↑.left←pNode
       ELSE
         q↑.right ← pNode
       END IF
       pNode↑.discrim ← RndCoord()
END
PROCEDURE 2DSearch(p,Rect,B)
BEGIN
       AVisSync(1)                                                (2)
       IF InsideRect(p↑.dat,rect) THEN   Output(p)
       Adjust (BH,BL,p↑.key)       /* Change bounding block  */
       AVisSync(1)
       IF p↑.left <> NIL AND BoundIntersectRegion(BL,Rect) THEN
         2DSearch(p↑.left,Rect,BL)
       IF p↑.right <> NIL AND BoundIntersectRegion(BH,rect) THEN
         2DSearch(p↑.right,Rect,BH)
END
```

รูปที่ 3-19 อัลกอริทึมการสร้างต้นไม้สองมิติโดยแกนที่เป็นคีย์ใช้ตัวเลขสุ่มและการค้นหา

3.3.5 ต้นไม้มัธยฐาน (Median Tree)[8]

ต้นไม้มัธยฐาน คือต้นไม้ที่ใช้วิธีการสร้างที่ปรับปรุงมาจากต้นไม้สองมิติ เนื่องจากในการสร้างต้นไม้แบบสองมิติ ลำดับของการใส่จุดลงไปในต้นไม้จะมีผลต่อความสูงของต้นไม้ ดังนั้นจึงมีการปรับปรุงวิธีการเลือก

[8] Friedman and Bentley, "An algorithm for finding best match in logarithmic expected time," ACM Transactions on Math. Software, 3(1977), p209-226.

โหนดที่ใส่เข้าไปในต้นไม้ โดยการเลือกจุดที่เป็นค่ามัธยฐานจากเซตของจุดที่พิจารณาเพื่อให้มีจำนวนโหนดในต้นไม้ย่อยแต่ละข้างเท่า ๆ กัน และปรับปรุงวิธีการเลือกแกนที่จะมาใช้เป็นคีย์ และใบของต้นไม้จะเป็นที่ฝากข้อมูล (bucket) ไม่ใช่ค่าของจุด

แนวคิดในการสร้างต้นไม้จะเลือกแกนที่จะใช้เป็นคีย์โดยพิจารณาเลือกแกนที่ระยะห่างระหว่างค่าที่มากที่สุดและค่าที่น้อยที่สุดมีค่ามากที่สุด แล้วหาค่ามัธยฐานจากแกนที่ใช้เป็นคีย์ แล้วเลือกจุดนั้นเป็นโหนดในต้นไม้ เมื่อมีจำนวนจุดเหลือน้อยกว่าขนาดของที่ฝากข้อมูลจึงกำหนดให้เป็นใบ ในที่นี้ใช้ที่ฝากข้อมูลมีขนาดเท่ากับ 4 เนื่องจากในงานวิจัยของ Friedman และ Bentley [9] ได้กำหนดค่าขนาดของที่ฝากข้อมูลที่เหมาะสมคือ 4 ถึง 32 จุด

การค้นหาจะเริ่มต้นเปรียบเทียบจากโหนดรากก่อน แล้วในแต่ละโหนด จะตรวจสอบว่าโหนดนั้นเป็นใบหรือไม่ ถ้าเป็นใบจะทดสอบว่าขอบเขตของใบนั้นอยู่ภายในพิสัยที่ต้องการค้นหาหรือไม่ ถ้าอยู่แสดงว่าไม่จำเป็นต้องตรวจสอบจุดในที่ฝากข้อมูลทีละจุด สามารถส่งค่าคืนได้เลย แต่หากไม่อยู่จะทำการตรวจสอบทีละจุด ในกรณีที่โหนดนั้นไม่ใช่ใบจะทำการทดสอบเหมือนกับต้นไม้สองมิติ

```
P แทน เซตของจุด =  { pi }  , 0 ≤ i ≤ N
l และ u แทนขอบเขตล่างและขอบเขตบนของจุด
Spreaddest (P;j) เป็นฟังก์ชันคืนค่าผลต่างระหว่างค่าที่มากที่สุดและค่าที่น้อยที่สุดของคีย์ที่ j ในที่นี้ค่า
j เป็น แกน x หรือแกน y
Adjust(BH,BL,p) เป็นกระบวนงานสำหรับปรับค่าขอบเขตตามค่าคีย์ของโหนด p ที่ค้นหาอยู่
PROCEDURE BuildMedTree(P;l,u)
BEGIN
      new(pNode)
      IF  Size(P) <= BUCKETSIZE   THEN RETURN(MakeTerminalNode ())
      maxspread ← 0
      FOR j ← 1  TO 2                                                    (1)
        AVisSync(l)
        IF  Spreaddest (P;j) > maxspread   THEN
          maxspread ← SPREADEST(P;j);    discrim ← j
        END IF
      NEXT j
      m ← (l+u) / 2
      Median(l,u,m,discrim)
      pNode↑.discrim ← discrim; pNode↑.cutval ← Pm,discrim
      pNode↑.loson ← BuildMedTree(P;l,m)
      pNode↑.hison ←BuildMedTree(P;m+1,u)
END
```

รูปที่ 3-20 อัลกอริทึมการสร้างต้นไม้มัธยฐานและการค้นหา

[9] Ibid.

```
PROCEDURE Median(l,u,m,discrim)
BEGIN
       r ← u
       WHILE (r > l )
         v ← p_{r,discrim}
         i ← l-1; j←r
         WHILE (p_{i,discrim} < v)
           AVisSync(1)                                              (2)
           i ← i+1
         END WHILE
         WHILE (p_{j,discrim} > v and j > l)
           AVisSync(1)                                              (3)
           j ← j-1
         END WHILE
         IF  i >= j  THEN   EXIT SUB
         swap(i,j)
         IF (i >=m) THEN r ← i-1
         IF (i <= m) THEN l ← i+1
         AVisSync(1)                                                (4)
       END WHILE
END
PROCEDURE  MedSearch(p,Rect,B)
BEGIN
       IF p is terminal node THEN
         IF  BoundContainRegion(B,Rect) THEN
           Output( all node in bucket)
         ELSE
           FOR i ← 1  TO  size(p)        /* check each node in bucket  */
             AVisSync(1)                                            (5)
             IF  InsideRect(p,Rect) THEN  Output (p)
           NEXT i
         END IF
       END IF
       Adjust (BH,BL,p)
       AVisSync(1)                                                  (6)
       IF  BoundIntersectRegion(BL,Rect) THEN  MedSearch(p↑.loson,Rect,BL)
       IF  BoundIntersectRegion(BH,Rect) THEN  MedSearch(p↑.hison,Rect,BH)
END
```

รูปที่ 3-20 อัลกอริทึมการสร้างต้นไม้มัธยฐานและการค้นหา (ต่อ)