บทที่ 2

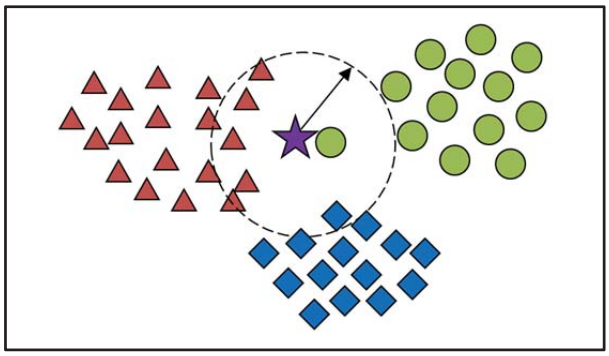
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 K-Nearest Neighbors

วิธีการนี้เป็นวิธีการในการจำแนกข้อมูลว่าใกล้เคียงกับข้อมูลในกลุ่มใดมากที่สุด โดยการหาข้อมูลตัวที่ใกล้เคียงกับข้อมูลตัวอย่างมากที่สุด จำนวน K ตัว เพื่อนำมาโหวตเป็นคำตอบว่าอยู่ใกล้กับข้อมูลในกลุ่มใดมากที่สุด วิธีการนี้ประสบความสำเร็จในการนำไปประยุกต์ใช้งานอย่างกว้างขวาง คำตอบที่เลือกออกมาจำนวน K ตัว สามารถที่จะทำการ Vote หรือ เฉลี่ยผลได้ ด้วยเหตุนี้วิธีการนี้จึงทนต่อ outlier ที่อาจส่งผลต่อผลของการจำแนกกลุ่มของข้อมูลได้เป็นอย่างดี ดังแสดงในภาพที่ 1

ภาพที่ 1

แสดงวิธีการของ K-Nearest Neighbors



จากภาพที่ 1 เป็นตัวอย่างการเลือกคำตอบจากวิธีการของ K-Nearest Neighbors จะเห็นได้ว่า เราสามารถที่จะตอบได้ว่าดาวอยู่ใกล้กับข้อมูลกลุ่มสามเหลี่ยมได้อย่างถูกต้องถึงแม้จะมีสี่เหลี่ยมขนมเปียกปูนและวงกลมรวมอยู่ด้วย ถ้าหากเราทำการเลือกตัวที่ใกล้ที่สุดเพียงตัวเดียว คำตอบจะเป็น วงกลม ที่เป็น outlier จะทำให้คำตอบที่ได้นั้นผิดไปจากความเป็นจริง ดังนั้นวิธีการนี้จึงให้ผลในการจำแนกข้อมูลได้ดีกว่าการตอบจากข้อมูลตัวที่ใกล้ที่สุดเพียงตัวเดียว

วิธีการนี้มีการนำไปประยุกต์ใช้งานและได้ผลดีกับการจำแนกข้อมูลในหลายๆ ด้าน เช่น การจดจำลายมือ (Lee, 1991) การจำแนกยีนส์ (Golub et al., 1999) การจำแนกสมองที่ทำงานผิดปกติ (Chaovalitwongse, Fan, & Sachdeo, 2007) และทางเหมืองข้อมูล (Weiss et al., 1999)

2.2 ค่าเฉลี่ยเคลื่อนที่ (Moving Average)

ค่าเฉลี่ยเคลื่อนที่เป็นวิธีการหนึ่งในการปรับให้เรียบ วิธีการนี้จะทำการเฉลี่ยข้อมูลโดยให้ค่าน้ำหนักของข้อมูลที่อยู่ใน moving window เท่ากัน วิธีการนี้จะให้ผลของแนวโน้มทิศทางของข้อมูลที่ไม่คลุมเครือซึ่งต่างจากเทคนิคอื่นๆ และสามารถที่จะนำไปทำการวิเคราะห์รูปแบบข้อมูล (pattern) ที่มีความไม่แน่นอนสูงได้

2.3 การวัดความคลาดเคลื่อนของการพยากรณ์

วิธีการวัดความคลาดเคลื่อนของการพยากรณ์ข้อมูลอนุกรม (series) ที่นิยมใช้ ได้แก่

2.3.1 Mean Absolute Error (MAE)

ค่า Mean Absolute Error คำนวณได้จากค่าเฉลี่ยของค่าสัมบูรณ์ของค่าความคลาดเคลื่อนแต่ละตัว หากมีการพยากรณ์ทั้งหมด N ช่วงเวลา ค่า MAE คำนวณได้จาก

$$MAE = \frac{1}{N}\sum_{t=1}^{N}|a_t - f_t| \qquad (2-1)$$

เมื่อ

a_t คือ ค่าจริงที่เกิดขึ้น ณ เวลา t

f_t คือ ค่าที่คาดการณ์ ณ เวลา t

ค่า MAE ยิ่งน้อย หมายถึง การพยากรณ์ยิ่งแม่นยำ

3.3.2 Root Mean Square Error (RMSE)

ค่า Root Mean Square Error คำนวณได้จากค่าเฉลี่ยของกำลังสองของค่าความคลาดเคลื่อนแต่ละตัว หากมีการพยากรณ์ทั้งหมด N ช่วงเวลา ค่า RMSE คำนวณได้จาก

$$RMSE = \frac{1}{N}\sum_{t=1}^{N}(a_t - f_t)^2 \qquad (2-2)$$

เมื่อ

a_t คือ ค่าจริงที่เกิดขึ้น ณ เวลา t

f_t คือ ค่าที่ค่าดการณ์ ณ เวลา t

ค่า RMSE ยิ่งน้อย หมายถึง การพยากรณ์ยิ่งแม่นยำ

2.3.3 Mean Absolute Percentage Error (MAPE)

ค่า Mean Absolute Percentage Error คำนวณได้จากค่าเฉลี่ยของค่าสัมบูรณ์ของค่าความคลาดเคลื่อนแต่ละตัวโดยคิดเป็นเปอร์เซ็นต์เทียบกับค่าที่เกิดขึ้นจริง หากมีการพยากรณ์ทั้งหมด N ช่วงเวลา ค่า MAPE คำนวณได้จาก

$$MAPE = \frac{1}{N}\sum_{t=1}^{N}\frac{|a_t - f_t|}{a_t} \qquad (2-3)$$

เมื่อ

a_t คือ ค่าจริงที่เกิดขึ้น ณ เวลา t

f_t คือ ค่าที่ค่าดการณ์ ณ เวลา t

ค่า MAPE ยิ่งน้อย หมายถึงการพยากรณ์ยิ่งแม่นยำ

2.4 Dynamic Programming

Dynamic Programming เป็นวิธีการที่ถูกคิดค้นขึ้นมาโดยนักคณิตศาสตร์ที่ชื่อ Richard Bellman ในปี 1950 (Bellman & Dreyfus, 1962) Dynamic Programming จะใช้ในการแก้ปัญหา optimization โดยมีลักษณะคล้ายกับการแก้ปัญหาแบบ Divide-and-Conquer โดยจะแบ่งปัญหาใหญ่ออกเป็นส่วนย่อยๆ แต่ละปัญหาย่อยนั้นจะต้องมีความสัมพันธ์ต่อเนื่องกัน (Recursive Function) การแบ่งปัญหาใหญ่เป็นปัญหาย่อยๆ ก็เพื่อหลีกเลี่ยงการคำนวณหาคำตอบของปัญหาที่ซ้ำๆกัน โดยจะแก้ปัญหาย่อยๆเหล่านั้นเพียงครั้งเดียว แล้วเก็บผลลัพธ์ที่ได้เอาไว้ หากพบว่าต้องการแก้ปัญหานั้นซ้ำอีกก็สามารถนำคำตอบที่เก็บไว้มาคำนวณต่อได้เลยไม่ต้องคำนวณใหม่ หลังจากนั้นจะนำผลลัพธ์ของปัญหาย่อยๆเหล่านั้นมารวมกันเพื่อแก้ปัญหาหลัก

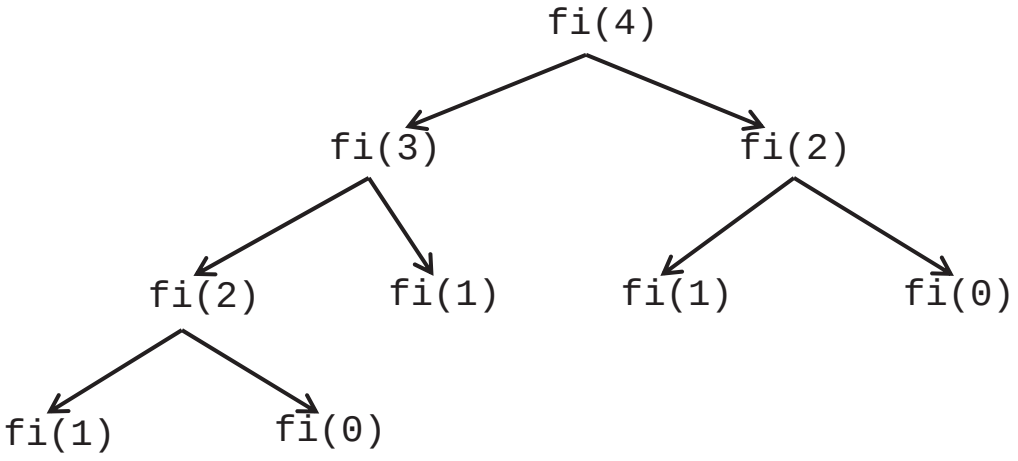
ตัวอย่างเช่น การหาเลข Fibonacci

เลข Fibonacci คือ ลำดับที่เริ่มต้นด้วย f_0 และ f_1 (ในที่นี้จะกำหนด $f_0 = 0$ และ $f_1 = 1$) จากนั้น f_n ตัวต่อไปคือ $f_n = f_{n-1} + f_{n-2}$ เมื่อ $n \geq 2$ สามารถเขียน code ได้ดังนี้

```
int fi (int n){
     if(n==0)
          return 0;
     if(n==1)
          return 1;
     return fi(n-1) + fi(n-2);
}
```

ถ้าสมมติว่าใส่ค่า $n = 4$ จะเกิดการทำงานดังนี้

ภาพที่ 2
แสดงการเรียกซ้ำของฟังก์ชั่น fi(4)



จากภาพที่ 2 จะเห็นได้ว่ามีการทำงานที่ซ้ำกันเกิดขึ้นแต่ถ้าเรียก fi(n) แล้วเก็บค่า fi(n-1) และ fi(n-2)ไว้ก่อนก็ไม่จำเป็นต้องคำนวณค่าต่างๆซ้ำ สามารถปรับปรุงโปรแกรมได้ดังนี้

```
int fi (int n){
     F[0] = 0;
     F[1] = 1;

     for(i=2; i<=n; i++)
          F[i] = F[i-1] + F[i-2];

     return F[n];
}
```

เวลาโปรแกรมนี้ทำงานก็จะใช้เวลาในการคำนวณลดลง

```
F[0] = 0;

F[1] = 1;

F[2] = F[1] + F[0] = 1;

F[3] = F[2] + F[1] = 2;

F[4] = F[3] + F[2] = 3;
```

2.5 วิธีการวัดความคล้ายของข้อมูลอนุกรมเวลา

ข้อมูลอนุกรมเวลา (Time Series Data) เป็นข้อมูลใด ๆ ที่ได้จากการเก็บค่าจุดข้อมูลอย่างต่อเนื่องตามลำดับของเวลาก่อนหลัง เช่น ข้อมูลความเร็ว ข้อมูลระดับน้ำทะเล ข้อมูลตลาดหุ้น ข้อมูลอุณหภูมิรายวัน ข้อมูลคลื่นหัวใจ เป็นต้น ในปัจจุบันข้อมูลอนุกรมเวลาได้เข้ามามีบทบาทที่สำคัญกับงานวิจัยในศาสตร์แขนงต่างๆ เช่น การรู้จำคำพูด (Speech Recognition) การเงิน (Finance) วิทยาการหุ่นยนต์ (Robotics) การแพทย์ (Medicine) การรู้จำอากัปกิริยาท่าทาง (Gesture Recognition) ชีวมาตร (Biometrics) ฯลฯ

การวัดความคล้ายของอนุกรมเวลามีวิธีที่เป็นที่รู้จักและใช้กันอย่างแพร่หลายอยู่สองวิธีใหญ่ๆ คือ วิธี Euclidian Distance และวิธี Dynamic Time Warping

2.5.1 Euclidian Distance

Euclidean Distance หรือ Euclidean Metric คือ ระยะทางระหว่างจุด 2 จุด จุดที่จะวัดนั้นอาจจะมีค่าจากหลายมิติหรือขนาดขึ้นอยู่กับรูปแบบของข้อมูล ซึ่งสามารถพิสูจน์หาค่าได้ด้วยทฤษฎีของ Pythagorean

<u>ระยะทางใน 1 มิติ</u>

ระยะทางระหว่างจุดสองจุดใน 1 มิติ ระหว่าง $P = [p_x]$ และ $Q = [q_x]$ สามารถหาได้จากสูตร

$$\sqrt{(p_x - q_x)^2} \qquad (2-4)$$

ระยะทางใน 2 มิติ

ระยะทางระหว่างจุดสองจุดใน 2 มิติ ระหว่าง $P = [p_x, p_y]$ และ $Q = [q_x, q_y]$ สามารถหาได้จากสูตร

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \qquad (2-5)$$

ระยะทางใน 3 มิติ

ระยะทางระหว่างจุดสองจุดใน 3 มิติ ระหว่าง $P = [p_x, p_y, p_z]$ และ $Q = [q_x, q_y, q_z]$ สามารถหาได้จากสูตร

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2} \qquad (2-6)$$

ระยะทางใน n มิติ

ระยะทางระหว่างจุดสองจุดใน n มิติ ระหว่าง $P = [p_1, p_2, \dots, p_n]$ และ $Q = [q_1, q_2, \dots, q_n]$ สามารถหาได้จากสูตร

$$\sqrt{(p_1 - q_2)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \qquad (2-7)$$

2.5.2 Dynamic Time Warping (DTW)

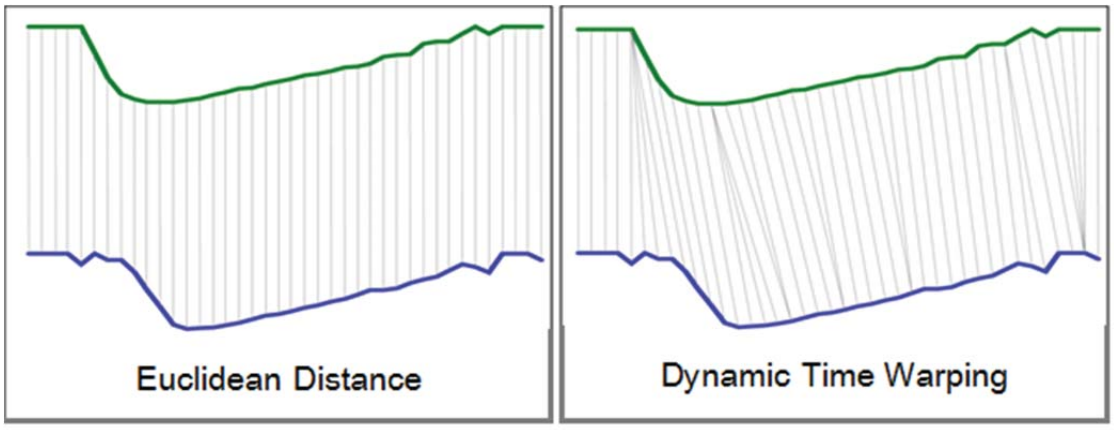
Dynamic Time Warping (Niels, 2004) เป็นวิธีที่ใช้สำหรับวัดความคล้ายระหว่างข้อมูลอนุกรมเวลา 2 ชุด โดยผลลัพธ์ที่ได้ จะให้ค่าระยะทางและวิถีการปรับแนว (Alignment) ที่ดีที่สุดระหว่างข้อมูลทั้งสอง ซึ่งสามารถยืด/หด ให้รองรับความแปรผันในแกนเวลาได้เป็นอย่างดี ในการคำนวณหาระยะทางจากวิธีการของ Dynamic Time Warping ได้นำเอาวิธีการของการโปรแกรมแบบพลวัต (Dynamic programming) เข้ามาช่วยลดเวลาที่ใช้ในการคำนวณ ดังตัวอย่าง Code ที่แสดงด้านล่าง

```
DTW (s[1..n], t[1..m]) {

  float cell[1..n+1, 1..m+1]
  int i, j,
  float cost, min


  cell [0, 0] := 0

  for i := 1 to m
    cell [0, i] := infinity

  for i := 1 to n
    cell [i, 0] := infinity



  for i := 1 to n

    for j := 1 to m
      cost:= d(s[i], t[j])
      min = minimum(cell [i-1, j  ],    // insertion
                    cell [i  , j-1],    // deletion
                    cell [i-1, j-1])    // match
      cell [i, j] := cost + min


  return cell [n, m]

}
```

Dynamic Time Warping เป็นวิธีที่มีการนำไปประยุกต์ใช้กับระบบรู้จำเสียงพูด (Speech Recognition) การรู้จำลายมือ (Handwriting Recognition) การจำแนก ประเภทดนตรี และการประมวลผลสัญญาณต่าง ๆเป็นต้น (Berndt & Clifford, 1994; Keogh & Pazzani, 2000; Kovacs-Vajna, 2000; Mongeau & Sankoff, 1990) โดยมีประสิทธิภาพในด้านความแม่นยำสูงสุด อย่างไรก็ตาม ข้อจำกัดหลักของวิธีนี้คือเวลาที่ต้องใช้ในการประมวลผลที่ค่อนข้างมาก

ภาพที่ 3

แสดงความแตกต่างระหว่างวิธีการของ Euclidean Distance และ Dynamic Time Warping



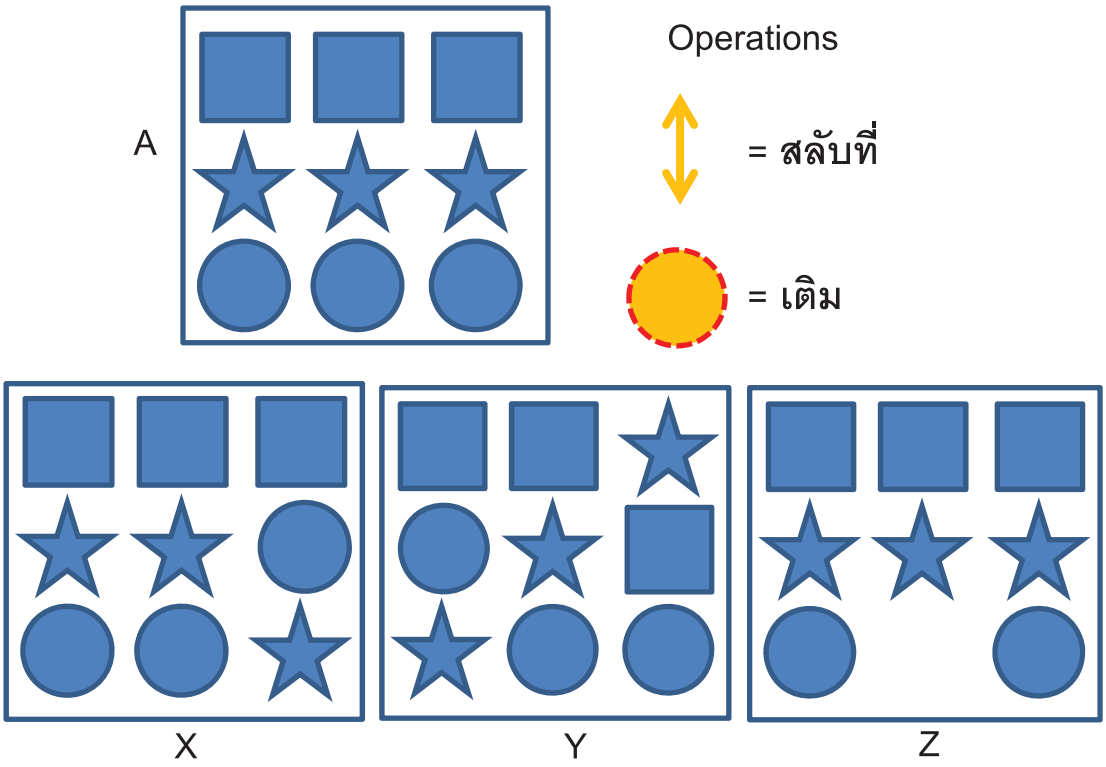
จากภาพที่ 3 จะเห็นได้ว่าวิธีการของ Dynamic Time Warping นั้นสามารถที่จะยืด/หด รองรับการแปรผันข้อมูลในแกนได้ดีกว่า Euclidean Distance

2.6 วิธีการวัดความคล้ายของข้อมูลลำดับ

วิธีการวัดความคล้ายของข้อมูลลำดับที่มีการคิดค้นขึ้นมามีหลากหลายวิธี ซึ่งวิธีการส่วนใหญ่ที่นำมาใช้นั้นจะมีแนวความคิดมาจากการ edit โดยใช้ operation ชุดเดียวกัน ทำให้สองสิ่งที่นำมาวัดความคล้ายนั้นเหมือนกัน แล้วคิดความคล้ายจาก operation ที่ใช้ ตัวอย่าง เช่น ถ้าหากเราจะหาว่ารูปใดคล้ายกับ รูป A มากที่สุดจาก 3 รูป โดยใช้ 2 operations คือ สลับที่ และ เติม

ภาพที่ 4

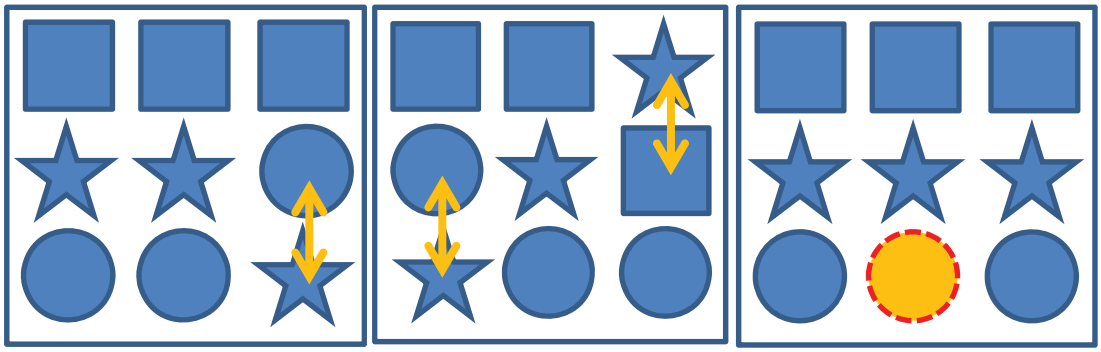
ภาพแม่แบบ (A), ภาพที่ต้องการวัดความคล้าย (X,Y,Z) และ operation สำหรับการ edit



จาก operations ที่กำหนดให้ สามารถที่จะ edit ทำให้รูป X, Y, Z นั้นกลายเป็นรูป A ได้ดังนี้

ภาพที่ 5

ภาพแสดงขั้นตอนของการ edit โดยใช้ operation ที่กำหนด



รูป X นั้น ใช้การสลับที่ 1 ครั้ง รูป Y ใช้การสลับที่ 2 ครั้ง ส่วนรูป Z ใช้การเติม 1 ครั้ง

ถ้าหากนับเพียงจำนวนของ operation ที่ใช้เพียงอย่างเดียวโดยไม่สนใจชนิดของ operation รูป Z

จะมีความคล้ายเท่ากับรูป X แต่ถ้าหากกำหนดให้ cost ของ การสลับที่เท่ากับ 1 และ cost ของการเติมเท่ากับ 3 รูป X จะคล้ายกว่ารูป Z ทันที

จากตัวอย่างแนวความคิดการวัดความคล้ายจากการ edit ด้วย operation กลุ่มเดียวกัน ให้สิ่งที่ต้องการวัดความคล้ายนั้นเหมือนกัน มีการนำไปประยุกต์ใช้กับการหาความคล้ายของสิ่งต่างๆ ในหลายๆ ด้านเช่น การหาความคล้ายของลำดับตัวอักษร, สาย DNA, รูปภาพ, เสียง เป็นต้น

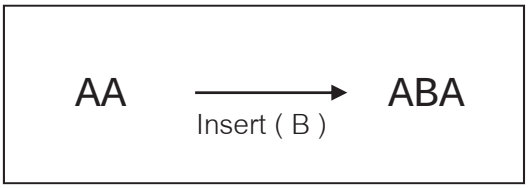
2.6.1 Edit Distance

Edit Distance (Morgan, H. L. 1970) เป็นวิธีการที่คิดค้นขึ้นในการเปรียบเทียบความคล้ายของสายอักขระสองสาย ซึ่งคำนวณค่าความคล้ายจากแต้มของแต่ละ operation ที่ใช้ในการทำให้สายอักขระสายหนึ่งเปลี่ยนไปเป็นสายอักขระอีกสายหนึ่ง โดยในตอนแรกได้นำไปใช้กับ complier สำหรับตรวจหาคำที่เหมือนกันจาก token ที่ผิดต่อมาในปี 1974 Wagner และ Fischer (Wagner & Fischer, 1974) ได้มีการปรับปรุงและเปลี่ยนวิธีการในการ edit และวิธีการในการคำนวณเพื่อให้มีการคำนวณที่ใช้เวลารวดเร็วขึ้น และเป็นที่รู้จักและนำไปใช้อย่างแพร่หลาย

วิธีการของ Edit Distance นั้นใช้หลักการการทำให้ sequence หนึ่งกลายเป็นอีก sequence หนึ่ง โดยจะใช้ 3 operation คือ insertion deletion substitution โดยจะนับความคล้ายจากการกระทำให้สอง sequence นี้เหมือนกัน โดยที่มีความยาวที่เป็นไปได้มากที่สุด

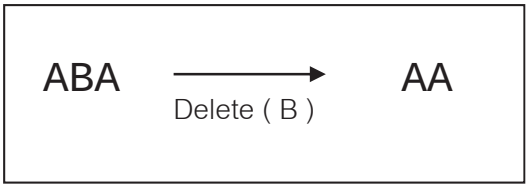
<u>Insertion Operation</u>

Operation นี้จะทำการแทรกตัวอักษรที่ต้องการลงไปยังตำแหน่งบนสายอักขระที่ต้องการ โดยจะแทรกตัวอักษรเข้าไปยังสายอักขระทีละตัว



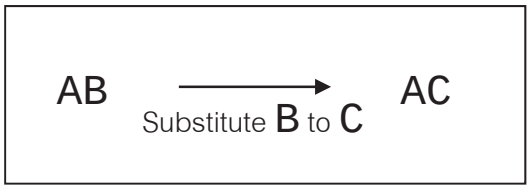
Deletion Operation

Operation นี้จะกระทำการตรงข้ามกับ insertion operation โดยการลบตัวอักษรที่ต้องการ โดยจะทำการลบตัวอักษรที่ต้องการทีละตัวในสายอักขระ



Substitution Operation

Operation นี้จะทำการเปลี่ยนตัวอักษรในสายอักขระ ให้เป็นตัวอักษรอื่นๆ ที่ต้องการ โดยสามารถเปลี่ยนตัวอักษรใดๆ บนสายอักขระได้ทีละตัวอักษร



ตัวอย่างเช่น ถ้าเราจะเปรียบเทียบ

ABCF และ ACBE

วิธีการของ Edit Distance สามารถเปลี่ยน ABCF ให้กลายเป็น ACBE ได้ โดยการลบ B ออกจาก ABCF จะได้ ACF, เพิ่ม B ไปตรงข้างหน้า F จะได้ ACBF, เปลี่ยน F ให้เป็น E จะได้ ACBE โดยทั้งหมดนี้มีการใช้ deletion ไป 1 ครั้ง, insertion ไป 1 ครั้ง, และ substitution ไป 1 ครั้ง

สำหรับคอมพิวเตอร์แล้วในการหาจำนวน operation ต่าง ๆ ที่ต้องใช้ทำการ edit คอมพิวเตอร์นั้นต้องทำการสร้าง sequence ที่เป็นไปได้ทั้งหมดขึ้นมา แล้วทำการเปรียบเทียบ ซึ่ง

ต้องทำการเปรียบเทียบเป็นจำนวนมาก และสิ้นเปลืองเวลาและหน่วยความจำในการคำนวณ แต่นักคณิตศาสตร์ชื่อ Richard Bellman ได้ทำการคิดค้นวิธีที่เรียกว่า Dynamic Programming ขึ้นมาเพื่อลดเวลาในการคำนวณลง Dynamic Programming ในที่นี้หมายถึงวิธีการหรือ Paradigm ไม่ได้หมายถึง Computer programming โดยมีวิธีการดังนี้

<u>วิธีการหา Distance จาก Edit Distance โดยใช้วิธีการ Dynamic programming</u>

ตัวอย่าง: หากต้องการหา Distance ของ S=[a,c,b] และ T=[a,b,c]

1. ทำการสร้างตารางสำหรับกระบวนการ Dynamic Programming

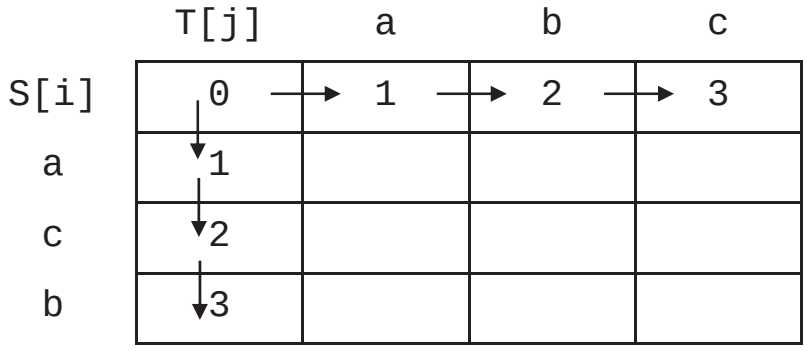
ทำการสร้างตารางที่มีขนาด (S+1) แถว (T+1) คอลัมน์

		a	b	c
a				
c				
b				

2. Initialize ค่าเริ่มต้นในตาราง

ก่อนการเติมค่าลงตาราง จะต้องทำการเติมค่าเริ่มต้นลงในตาราง โดยทำตาม recurrent relation ดังนี้

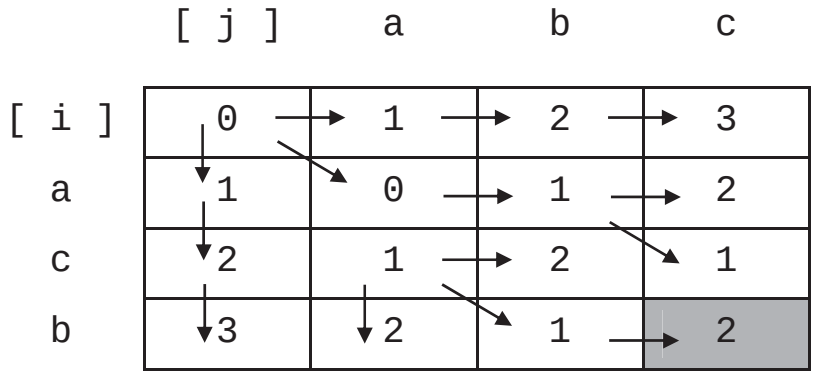
```
for i from 0 to m
     cell[i, 0] := i
for j from 0 to n
     cell[0, j] := j
```



3. Fill in Cell เติมค่าที่เหลือลงในช่องว่าง

ทำการเติมค่าลงในช่องตารางที่เหลือ ตาม recurrence relation ดังนี้

```
if s[i] = t[j] then
   cell[i, j] := cell [i-1, j-1]
else
   cell[i, j] := minimum
           (
             cell [i-1,  j ] + cost of insertion,
             cell [ i , j-1] + cost of deletion,
             cell [i-1, j-1] + cost of substitution
            )
```



โดยในที่นี้จะทำการกำหนด cost of insertion และ cost of deletion เท่ากับ 1 แต้ม และ cost of substitution เท่ากับ 2 แต้ม และทำการเติมลูกศรตามทางเดินที่ได้ทำการเลือกเดินเอาไว้

4. Trace back

หลังจากเติมค่า และเติมลูกศรลงในตารางจนครบ จะเห็นได้ว่ามีการเดินตามลูกศร → ↓ และ ↘ ซึ่งลูกศร → และ ↓ จะหมายความว่ามีการ edit ตัวอักษรที่ตำแหน่งนั้นด้วย operation และผลของการหา Distance จะอยู่ในช่องมุมขวาล่าง โดยจากตัวอย่างด้านบนจะได้ distance เท่ากับ 2 หมายความว่า มีแต้มจากการ edit ด้วย operation รวมทั้งหมด 2 แต้ม

2.6.2 Run-Length Encoded Edit Distance

Run-Length Encode คือ การลดขนาดของข้อมูลให้มีขนาดเล็กลง โดยการนับจำนวนที่ซ้ำของข้อมูลและแทนด้วยจำนวนตัวเลขที่นับได้ ตามด้วยรหัสของข้อมูลตัวนั้นที่มีการซ้ำกัน ข้อมูลที่มีค่าซ้ำๆ กัน จะถูกเก็บเป็นรหัสแทน และบอกถึงจำนวนของข้อมูลที่ซ้ำกันว่ามีกี่จำนวน ทำให้ลดจำนวนของข้อมูลที่ซ้ำกันลงไป เป็นหนึ่งในวิธีการทำ data compression โดยมีการนำไปใช้กับหลายๆ ข้อมูลเช่น รูปภาพแบบ bitmap pixel, เส้น, icon, animation และ การทำดัชนีคำ (Lemire, Kaser, & Aouiche, 2009)

ตัวอย่างเช่น ถ้าหากมีข้อมูลดังภาพที่ 6 เราสามารถที่จะใช้วิธีการของ Run-Length Encode เพื่อช่วยลดปริมาณของข้อมูลที่ต้องการเก็บได้ โดยจะได้ผลลัพธ์คือ **3A 6B 4C 4B 2C 5B 6A**

ภาพที่ 6
ตัวอย่างข้อมูลที่เกิดซ้ำๆกัน

A A A B B B B B B C C C C B B B B C C B B B B B A A A A A A

ในปี 1992 Bunke และ Csirik ได้คิดค้นวิธีการหา Edit Distance จาก Run-Length Encode (Bunke & Csirik, 1992) เพื่อให้ได้ผลลัพธ์ที่เท่ากับการหา Edit Distance จากข้อมูลที่ยังไม่ได้ทำ Run-Length Encode และมีการพัฒนาวิธีการให้มีการคำนวณที่เร็วขึ้นกว่าเดิมในปี 1995 (Bunke & Csirik, 1995) โดยใช้เวลาในการคำนวณเท่ากับความยาวของจำนวนข้อมูลที่ผ่านกระบวนการ Run-Length Encode โดยหนึ่งหน่วยของข้อมูลนั้นจะประกอบด้วย ความยาวและรหัสข้อมูล

ต่อมาในปี 2004 ได้มีการพัฒนาขั้นตอนวิธี (algorithm) ที่สามารถทำการคำนวณค่า Longest Common Subsequence ของ Run-Length Encode ในรูปแบบของการคำนวณแบบพร้อมกัน (parallelism) ได้ (Freschi & Bogliolo, 2004) โดยใช้เวลาในการคำนวณ Run-Length Encode X (ที่มี M characters m run) และ Y (ที่มี N characters n run) เท่ากับ $O(mN + nM - mn)$

2.6.3 Event sequence

Event คือข้อมูลชนิดหนึ่งที่เกิดขึ้นรอบๆตัวเรา โดยข้อมูลชนิดนี้จะประกอบด้วยชนิดของเหตุการณ์ที่เกิดขึ้น (event) และเวลาที่เกิดขึ้น (time) เช่น การแจ้งเตือนเหตุการณ์, การกระทำระหว่างมนุษย์กับส่วนติดต่อผู้ใช้ (user interface), การเกิดการระบาดของโรค, การเข้าใช้เวปไซต์ เป็นต้น

ข้อมูล event สามารถที่จะเขียนให้อยู่ในรูปของลำดับที่เรียงตามเวลาที่เกิดขึ้นได้ดังนี้ $E = (e_1, t_1), (e_2, t_2), \dots, (e_n, t_n)$

ในปี 1997 Mannila และ Ronkainen ได้เสนอวิธีการวัดความคล้ายของ Event sequence ขึ้น (Mannila & Ronkainen, 1997) โดยใช้ 3 operations ในการทำการ edit คือ

1. Insertion operation (การแทรก)
2. Deletion operation (การลบ)
3. Moving operation (การย้ายที่)

และใช้ dynamic programming ช่วยในการลดเวลาที่ใช้ในการคำนวณ โดยมี recurrence relation ดังนี้

$$
\begin{aligned}
r(0,0) &= 0 \\
r(i,0) &= r(i-1,0) + w(e_i) \\
r(0,j) &= r(0,j-1) + w(f_j) \\
r(i,j) &= min \begin{cases} r(i-1,j) + w(e_i) \\ r(i,j-1) + w(f_j) \\ r(i-1,j-1) + k(i,j) \end{cases} \\
k(i,j) &= \begin{cases} |t_i - u_j| & \text{if } e_i = f_j \\ w(e_i) + w(f_j) & \text{if } e_i \neq f_j \end{cases}
\end{aligned}
\qquad (2-8)
$$

เมื่อ

$w(e_i)$ และ $w(f_j)$ cost ของการ insert และ delete ตามลำดับ

2.6.4 Timed String

Timed String (Dobrišek, Zibert, Paveši, & Miheli, 2009) เป็นวิธีการในการใช้หาความคล้ายระหว่าง sequence โดยที่แต่ละหน่วยข้อมูลของ sequence จะประกอบด้วยข้อมูลในสองมิติ คือ มิติของเวลา และมิติของ symbol วิธีการของ Timed String ได้นำวิธีการของ Edit Distance มาทำการปรับปรุงเพิ่มเติมในส่วนของการคิด cost ของ operation ใหม่ โดยมีการเพิ่มการคิด cost ในส่วนการเปรียบเทียบเวลาด้วย ซึ่งทำให้สามารถที่จะทำการวัดความคล้ายของ sequence ในสองมิติได้ ทั้งมิติของเวลา และมิติของ symbol ทำให้การวัดนั้นมีความละเอียดเพิ่มขึ้น และสามารถเปรียบเทียบความคล้ายได้ดียิ่งขึ้น

วิธีการของ Timed String นั้นไม่ได้แตกต่างกับ Edit Distance มากนัก เพียงแต่มีการเพิ่มการคำนวณ cost ของเวลาเข้าไปในการหา cost ด้วย โดยในวิธีการของ Timed String ได้กำหนดให้หนึ่งหน่วยข้อมูลใน sequence ประกอบด้วย (Symbol, Start time, End time) เรียกว่า Timed Symbol และจะมีรูปข้อมูลที่ถูกกำหนดขึ้นมาเป็นพิเศษเพื่อช่วยในการคำนวณความคล้าย คือข้อมูลที่มีขนาดเป็นศูนย์ (Null Symbol) ซึ่งจะแทนที่ symbol ด้วย Empty Symbol เขียนแทนด้วยสัญลักษณ์ ε นั่นคือ (ε, Start time, End time) โดย Null Symbol นี้จะมีลักษณะพิเศษคือขนาดจะเป็นศูนย์ ซึ่งหมายความว่า หากนำ Null Symbol นี้ไปต่อกับ sequence ใดๆ ขนาดของ sequence ก็จะยังคงมีขนาดเท่าเดิม

วิธีการของ Timed String จะใช้วิธีการของ Dynamic Programming เช่นเดียวกับ Edit Distance โดยมีการกำหนด Recurrence Equation ในขั้นตอน Initialized ดังนี้

$$\begin{aligned} d(0,0) &= c(\varphi_0, \omega_0) \\ d(i,0) &= d(i-1,0) + c(\varphi_i \rightarrow a_i) \\ d(0,j) &= d(0,j-1) + c(b_i \rightarrow \omega_j) \end{aligned} \qquad (2-9)$$

เมื่อ φ_i และ ω_j คือ Null Symbol ณ ตำแหน่ง i และ j , ฟังก์ชั่น $c(\varphi_0, \omega_0)$ คือ cost ของการปรับเวลาเริ่มต้นของสอง sequence, ฟังก์ชั่น $c(\varphi_i \rightarrow a_i)$ คือ cost ของการลบ Timed Symbol, ฟังก์ชัน $c(b_i \rightarrow \omega_j)$ คือ cost ของการแทรก Timed Symbol

สำหรับ Recurrence Equation ในขั้นตอน Fill in มีดังนี้

$$d(i,j) = \min \begin{cases} d(i-1,0) + c(\varphi_i \rightarrow a_i), \\ d(0,j-1) + c(b_i \rightarrow \omega_j), \\ d(i-1,j-1) + m(a_i, b_j) \end{cases} \quad (2-10)$$

$$m(a,b) = \{\beta \times p(start(a), end(a), start(b), end(b))\} + \{(1-\beta) \times q(a,b)\}$$

เมื่อ $m(a_i, b_j)$ คือ ผลรวมของ cost ความแตกต่างของเวลา $p(start(a), end(a), start(b), end(b))$ และ Event $q(a,b)$ โดยทำการถ่วงน้ำหนัก ซึ่งผู้ใช้สามารถที่จะกำหนดค่าน้ำหนักที่ใช้ถ่วงได้เอง

เมื่อ Fill in ตารางจนครบแล้ว ค่าความคล้ายที่เป็นผลลัพธ์ระหว่าง 2 Timed String จะอยู่ในช่องมุมขวาล่าง จาก recurrence equation จะเห็นได้ว่า มีการคิด cost ของเวลา และ cost ของ Event แล้วนำไป weight เพื่อให้ได้ cost ของการ edit ตรงนี้เป็นส่วนที่เพิ่มเติมขึ้นมาจากวิธีการของ Edit Distance โดยทั่วไป