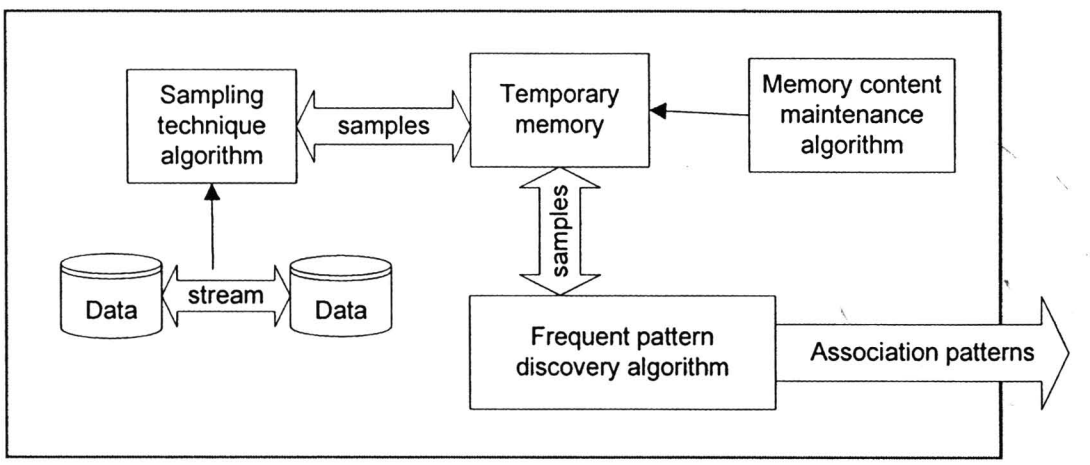
บทที่ 2
การออกแบบและพัฒนาโปรแกรม

2.1 การออกแบบวิธีการค้นพบโดยประมาณของรูปแบบที่ปรากฏบ่อย

แนวทางการดำเนินงานของโครงการวิจัยนี้ (รูปที่ 2.1) จะประกอบด้วยการออกแบบและพัฒนาอัลกอริทึมในการสุ่มเลือกข้อมูลตัวแทนจากสตรีม (sampling technique algorithm) การคัดเลือกและ maintain ข้อมูลตัวแทน (memory content maintenance algorithm) และการค้นพบรูปแบบที่ปรากฏบ่อยจากกลุ่มข้อมูลตัวแทน (frequent pattern discovery algorithm)



รูปที่ 2.1 กรอบของงานออกแบบวิธีการค้นพบรูปแบบที่ปรากฏบ่อยในข้อมูลสตรีม

สมมติฐานของงานวิจัยนี้เป็นการค้นพบรูปแบบที่ปรากฏบ่อยจากข้อมูลสตรีม โดยที่สตรีมอาจจะมาจากแหล่งกำเนิดข้อมูลที่แตกต่างกัน ทำให้ลักษณะการกระจายของข้อมูลอาจจะแตกต่างกันมาก วิธีการออกแบบอัลกอริทึมในลักษณะของการประมาณ (approximation) จะใช้วิธีการสุ่มเลือก เป็นพื้นฐานหลัก แต่จะลำเอียงการสุ่มตามความหนาแน่นในการกระจายตัวของข้อมูล ในงานวิจัยนี้กำหนดข้อมูลให้อยู่ในลักษณะของเวกเตอร์ และใช้เทคนิคการเลื่อนกรอบหน้าต่าง (window sliding) ที่มีขนาดคงที่ไปบนแกนข้อมูลเพื่อวัดความหนาแน่นของกลุ่มข้อมูลย่อย (หมายถึงข้อมูลทั้งหมดที่อยู่ในกรอบหน้าต่างเดียวกัน) ข้อมูลกลุ่มย่อยที่มีความหนาแน่นถึงเกณฑ์ที่กำหนดจะถูกคัดเลือกเก็บไว้ในเนื้อที่หน่วยความจำชั่วคราวที่เรียกว่า reservoir (Vitter, 1985; Kerdprasop, Kerdprasop, & Sattayatham, 2005) ข้อมูลใน reservoir จะเป็นข้อมูลที่ถูกคัดเลือกไปใช้ในการค้นพบรูปแบบที่ปรากฏบ่อย

การคำนวณความหนาแน่นของข้อมูลด้วยเทคนิคการเลื่อนกรอบหน้าต่าง แสดงขั้นตอนการทำงานได้ดังรูปที่ 2.2 ข้อมูลที่ผ่านกระบวนการนี้จะถูกแปลงรูปแบบและลดจำนวนด้วยการแสดงเฉพาะข้อมูลตัวแทน (ใช้จุดกึ่งกลางของกรอบหน้างต่างเป็นตัวแทนข้อมูลทั้งหมดในกรอบหน้าต่างนั้น) จากนั้น

เมื่อถึงขั้นตอนการสุ่มตามความหนาแน่น ผู้ใช้จะระบุค่าความหนาแน่นขั้นต่ำของข้อมูลในแต่ละกรอบหน้าต่าง เฉพาะกรอบหน้าต่างที่มีค่าความหนาแน่นถึงเกณฑ์จะถูกนำไปใช้ในขั้นตอนการสุ่ม

Algorithm Window sliding

Input: a set of data points represented as vectors

Output: a new set of transformed data points annotated with density value

% Initialize windows

(1) Interact with user to obtain dimension value

(2) Generate window grid of size *W* along dimension axes

% Count density

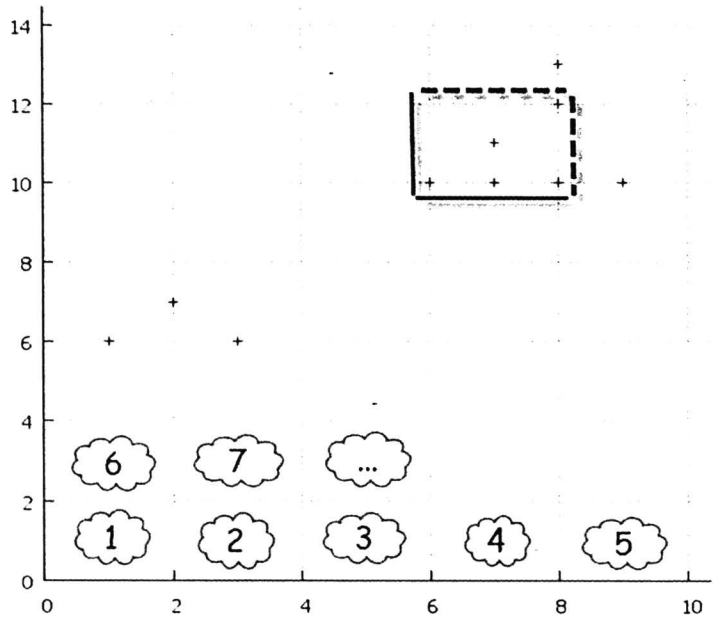
(3) Sequential move on each window and count number of data points, *N*, in the window

(4) Record a list of window's central point and its *N* value in a file *F*

(5) Return *F* as a set of transformed data

รูปที่ 2.2 อัลกอริทึมการเลื่อนกรอบหน้าต่างเพื่อคำนวณความหนาแน่นของข้อมูล

ถ้ากำหนดเวกเตอร์ของข้อมูลตัวอย่างจำนวน 10 ข้อมูลขนาดสองมิติ (นั่นคือ แต่ละข้อมูลประกอบด้วยสองไอเท็ม ในตัวอย่างนี้ข้อมูลแต่ละรายการแทนจุด โดยไอเท็มแรกเป็นตำแหน่งของจุดในแนวแกน x และไอเท็มที่สองเป็นตำแหน่งของจุดในแนวแกน y) เป็นดังนี้ <2,7>, <3,6>, <1,6>, <9,10>, <8,12>, <7,11>, <6,10>, <7,10>, <8,13>, <8,10> รูปที่ 2.3 แสดงแผนภาพของการกำหนดกรอบหน้าต่าง (แสดงด้วยเส้นประบาง) และแสดง Stream ของกรอบหน้าต่างด้วยหมายเลข 1, 2, 3, ..., 35 กรอบหน้าต่างหมายเลข 1 จะครอบคลุมขอบเขตของพิกัด <x,y> ในช่วงค่า <0..1.99, 0..1.99> โดยไม่รวมขอบเขตบนที่พิกัด x=2 และ y=2 และกรอบหน้าต่างหมายเลข 2 จะครอบคลุมขอบเขตของพิกัด <x, y> ในช่วงค่า <2..3.99, 0..1.99> เรียงเป็นลำดับเช่นนี้ไปจนถึงกรอบหน้าต่างหมายเลข 35



รูปที่ 2.3 การกำหนดกรอบหน้าต่างและขอบเขตการนับจำนวนข้อมูลในแต่ละกรอบหน้าต่าง

ในรูปที่ 2.3 แสดงขอบเขตของกรอบหน้าต่างหมายเลข 29 ที่อยู่ในช่วงพิกัด <6..7.99, 10..11.99> โดยใช้เส้นประหนาแสดงให้เห็นว่าขอบเขตบนจะไม่รวมพิกัด y=12 และขอบเขตด้านขวาจะไม่รวมพิกัด x=12 ดังนั้นข้อมูลที่อยู่ภายในขอบเขตของกรอบหน้าต่างที่ 29 จะประกอบด้วยข้อมูล 3 ตัว คือ <6,10>, <7,10> และ <7,11> กรอบหน้าต่างนี้มีตำแหน่งกึ่งกลางอยู่ที่พิกัด <7,11> ดังนั้นข้อมูลทั้ง 3 ตัวจะถูกเปลี่ยนรูปแบบการเก็บเป็น {<7,11>, 3} ซึ่งจะเป็นรูปแบบที่ย่อลงกว่าการเก็บจุดข้อมูลที่แท้จริง และจะช่วยให้การเก็บข้อมูลใช้เนื้อที่น้อยลงด้วย จากกรอบหน้าต่างทั้งหมดที่มีข้อมูลปรากฏอยู่สรุปได้ดังรูปที่ 2.4

Window 16:
ขอบเขต: <0..1.99, 6..7.99>
ข้อมูล: <1,6>
ข้อมูลที่แปลงแล้ว: {<1,7>,1}

Window 17:
ขอบเขต: <2..3.99, 6..7.99>
ข้อมูล: <2,7>, <3,6>
ข้อมูลที่แปลงแล้ว: {<3,7>,2}

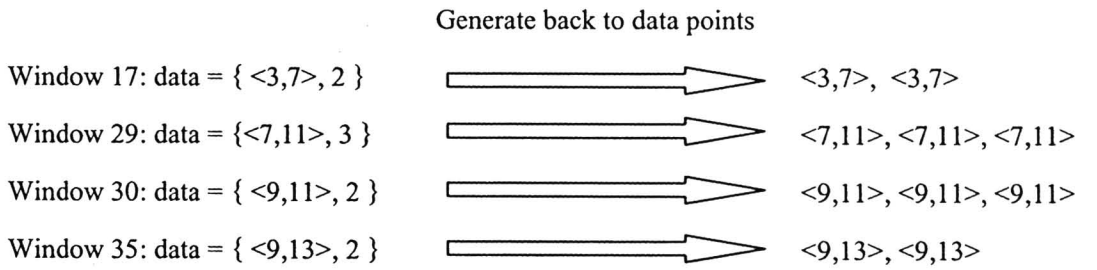
Window 29:
ขอบเขต: <6..7.99, 10..11.99>
ข้อมูล: <6,10>, <7,10>, <7,11>
ข้อมูลที่แปลงแล้ว: {<7,11>,3}

Window 30:
ขอบเขต: <8..9.99, 10..11.99>
ข้อมูล: <8,10>, <9,10>
ข้อมูลที่แปลงแล้ว: {<9,11>,2}

Window 35:
ขอบเขต: <8..9.99, 12..13.99>
ข้อมูล: <8,12>, <8,13>
ข้อมูลที่แปลงแล้ว: {<9,13>,2}

รูปที่ 2.4 ข้อมูลสรุปของกรอบหน้าต่างที่ 16, 17, 29, 30 และ 35

อัลกอริทึมการสุ่มข้อมูลตามความหนาแน่น จะใช้ข้อมูลที่ผ่านการแปลงรูปแบบและวัดความหนาแน่นด้วยอัลกอริทึม Window sliding โดยผู้ใช้จะเป็นผู้กำหนดเกณฑ์ขั้นต่ำของค่าความหนาแน่นในแต่ละกรอบหน้าต่าง เฉพาะกรอบหน้าต่างที่มีความหนาแน่นถึงเกณฑ์เท่านั้นที่จะถูกคัดเลือกไปใช้ โดยก่อนที่จะเริ่มกระบวนการคัดเลือกจะต้องมีการแปลงรูปแบบข้อมูลกลับไปเป็นจุดข้อมูล วิธีการแปลงจะใช้จุดกึ่งกลางกรอบหน้าต่างเป็นพิกัดของข้อมูล จากนั้นสร้างพิกัดข้อมูลนั้นซ้ำให้มีจำนวนเท่ากับความหนาแน่นของกรอบหน้าต่างนั้นๆ เช่นถ้ากำหนดความหนาแน่นขั้นต่ำเป็น 2 จากจำนวนข้อมูลตัวอย่าง 10 ข้อมูล จะมีข้อมูลกลุ่มย่อยที่ตรงตามเกณฑ์แสดงได้ดังรูปที่ 2.5



รูปที่ 2.5 การแปลงข้อมูลกลับเป็นเวกเตอร์ที่แทนจุดข้อมูล

ข้อมูลที่ถูกแปลงกลับเหล่านี้จะถูกนำไปผ่านอัลกอริทึมสุ่มข้อมูล (รูปที่ 2.6) ที่ใช้เทคนิคการสุ่ม โครงสร้างการเก็บข้อมูล และวิธีการ maintain ข้อมูลที่แตกต่างกันดังนี้

- Density-biased Reservoir + Hashing ข้อมูลที่มีความหนาแน่นถึงเกณฑ์ จะถูกเก็บลงโครงสร้างข้อมูล Reservoir ที่มีขนาดจำกัด และการเข้าถึงเนื้อที่เก็บข้อมูลใช้เทคนิคแฮชชิงด้วยฟังก์ชัน (n mod r)+1 เมื่อ n คือลำดับที่ของข้อมูล และ r คือ จำนวนเนื้อที่ใน Reservoir แต่ละเนื้อที่จะเก็บข้อมูลได้หนึ่งตัว ดังนั้นถ้าเกิดการชนกันของข้อมูลที่ผ่านการคำนวณด้วยฟังก์ชันแฮช ข้อมูลเดิมใน Reservoir จะถูกข้อมูลใหม่บันทึกทับ
- Density-biased Reservoir + Simple random sampling ข้อมูลที่มีความหนาแน่นถึงเกณฑ์ จะถูกเก็บลงโครงสร้างข้อมูล Reservoir ที่มีขนาดจำกัด โดยการเลือกเก็บข้อมูลลง Reservoir จะใช้วิธีการสุ่มอย่างง่าย
- Density-biased Reservoir + Rejection sampling ข้อมูลที่มีความหนาแน่นถึงเกณฑ์ จะถูกเก็บลงโครงสร้างข้อมูล Reservoir ที่มีขนาดจำกัด โดยการเลือกเก็บข้อมูลลง Reservoir จะใช้วิธีการสุ่มสองครั้งด้วยเทคนิค Rejection sampling การสุ่มครั้งแรกเป็นการสุ่มเลือกข้อมูล การสุ่มครั้งที่สองเป็นการสุ่มตัวเลข uniform ที่มีค่าอยู่ระหว่าง [0..1] จากนั้นเทียบค่า uniform ที่สุ่มได้ว่าอยู่ภายในช่วงที่กำหนดหรือไม่ (เช่น กำหนดช่วง [0.3..0.7]) ถ้าค่าไม่อยู่ภายในช่วงจะทิ้งข้อมูลนั้นไป (เรียกว่า reject) แต่ถ้าค่าอยู่ภายในช่วงจะเก็บข้อมูลนั้นลง Reservoir กระบวนการจะวนซ้ำจนกว่า Reservoir จะเต็ม

Algorithm Density-biased sampling

Input: a set of high density data generated back from the Window sliding algorithm
Output: a new set of data samples

(1) Display GUI to obtain a desired sampling technique choice from user
(2) If choice = 'Density-biased Reservoir+Hashing'
(3) Then Interactive with user to obtain reservoir size
(4) Hash each data point to store in a reservoir R
(5) If collision occurs, the stored data is replaced with a new one
(6) Repeat steps 4-5 until there is no more data point, and return R as output
(7) If choice = 'Density-biased Reservoir+Simple Random Sampling'
(8) Then Interact with user to obtain bin size
(9) Randomly select data point to store in a reservoir R
% random sampling without replacement
(10) Repeat step 9 until R is full, and return R as an output
(11) If choice = 'Density-biased Reservoir+Rejection Sampling'
(12) Then Interact with user to obtain bin size and interval I, I $\in$ [0.0..0.5]
(13) Randomly select data point D *% sampling without replacement*
(14) Generate a uniform random number U from the range [0.0 .. 1.0]
(15) If U is within the range [0.5-I .. 0.5+I], then store D in R
(16) Otherwise, reject and discard D
(17) Repeat steps 13-16 until R is full, and return R as an output

รูปที่ 2.6 อัลกอริทึมการสุ่มข้อมูลตามความหนาแน่นและการ maintain ข้อมูล

ข้อมูลสตรีมที่ผ่านขั้นตอนการสุ่มตามความหนาแน่น จะถูกส่งต่อไปยังกระบวนการค้นพบรูปแบบที่ปรากฏบ่อย ที่ใช้ขั้นตอนการทำงานเช่นเดียวกับอัลกอริทึม Apriori (Agrawal & Srikant, 1994)

2.2 การค้นพบรูปแบบที่ปรากฏบ่อยด้วยภาษาฮาสเกล

ภาษาฮาสเกลมีลักษณะเป็น pure functional language นั่นคือการทำงานของฟังก์ชันจะไม่เกิดผลข้างเคียงต่อฟังก์ชันอื่น การเขียนฟังก์ชันจะใช้วิธีการระบุรูปแบบอินพุทและเอาท์พุทในลักษณะของแพทเทิร์น และใช้วิธีการเรียกตัวเองซ้ำในลักษณะของ recursive เมื่อต้องการวนลูป ดังตัวอย่างในรูปที่ 2.7

```
-- .....
-- Fibonacci function in Haskell
   fib :: Int -> Int          -- a function takes one Integer and returns an Integer
   fib 0 = 0                  -- pattern 1: input is 0, output = 0
   fib 1 = 1                  -- pattern 2: input is 1, output = 1
   fib n = fib (n-2) + fib (n-1)   -- pattern 3: input is an integer other than 0 and 1,
                              --          then output = fib(n-2) + fib(n-1)
-- .....
```

รูปที่ 2.7 ตัวอย่างโปรแกรมภาษาฮาสเกลเพื่อคำนวณค่าไฟโบนาซซี

ภาษาฮาสเกลกำหนดรูปแบบชนิดข้อมูลอย่างเข้มงวด โดยห้ามนำข้อมูลต่างชนิดมากระทำกัน จากโปรแกรมตัวอย่างในรูปที่ 2.7 ข้อความ "fib:: Int -> Int" เป็นการประกาศว่าฟังก์ชันชื่อ "fib" รับข้อมูลเข้าเป็นชนิด integer (หรือ Int) และให้เอาท์พุทของฟังก์ชันเป็นค่า integer ข้อความดังกล่าวจึงถือเป็นส่วนประกาศชื่อและชนิดของฟังก์ชัน (ข้อความหลังเครื่องหมาย "--" เป็น comment หรือหมายเหตุของโปรแกรม) บรรทัดต่อจากนั้นเป็นข้อความ "fib 0 = 0" เป็นการระบุแพทเทิร์นว่าถ้าอินพุทเป็นค่าศูนย์ ฟังก์ชันจะให้ผลลัพธ์เป็นค่าศูนย์เช่นเดียวกัน ในแพทเทิร์นต่อมาเป็นคำสั่ง "fib 1 = 1" เป็นการระบุแพทเทิร์นว่าถ้าอินพุทเป็นค่าหนึ่ง ฟังก์ชันจะให้ผลลัพธ์เป็นค่าหนึ่งเช่นเดียวกัน และในคำสั่งสุดท้ายระบุเป็นแพทเทิร์น n ใดๆที่ไม่ใช่ 0 หรือ 1 ว่า "fib n = fib(n-2) + fib(n-1)" เช่น fib 2 จะได้ค่าจากผลบวกของ fib(0) และ fib(1) เป็นต้น คำสั่งสุดท้ายนี้ใช้ลักษณะของ recursive function เพื่อวนทำงานซ้ำ

การเขียนโปรแกรมค้นพบรูปแบบที่ปรากฏบ่อยด้วยภาษาฮาสเกล แสดงได้ดังรูปที่ 2.8

```
patternSet :: [Set Int]
patternSet =[Set.singleton x | x<-[1..9]]

sumi::Set Int->[Set Int]->Int
sumi s [] =0
sumi s (y:ys) |(Set.isSubsetOf s y)= 1+(sumi s ys)  |otherwise = (sumi s ys)

-- listC is a function to find candidate itemset
listC ::Int->[(Set Int,Int)]
listC 1=[let n=(sumi s dataB) in (s,n)  |s<-patternSet]
listC n=[let n=(sumi s dataB) in (s,n) |s<- Set.toList(listC' n)]

listC' :: Int->Set(Set Int)
listC' 2=Set.fromList [(Set.union x y) |x<-(listL' 1), y<-(listL' 1), x/=y]
listC' n=Set.fromList [(Set.union x y) |x<-(listL' (n-1)),  y<-(listL' (n-1)), x/=y,
                                        (Set.size(Set.union x y))==n]

-- listL is a function to find large (or frequent) itemset
listL ::Int->[(Set Int,Int)]
listL n=[(x,y)|(x,y)<-listC n, y>=minS]

listL'::Int->[Set Int]
 listL' n =[x|(x,_)<-listL n]
```


รูปที่ 2.8 โปรแกรมภาษาฮาสเกลเพื่อการค้นพบรูปแบบที่ปรากฏบ่อย

2.3 การพัฒนาโปรแกรมต้นแบบด้วยภาษาเออแลง

จากการทดสอบแนวทางการโปรแกรมเชิงฟังก์ชัน ในงานการค้นพบรูปแบบที่ปรากฏบ่อย พบว่าภาษาเชิงฟังก์ชันมีความเหมาะสมกับงานที่จะต้องระบุรูปแบบแพทเพิร์นเป็นจำนวนมาก ซึ่งภาษาในแนวทางนี้จะช่วยในลดเวลาในการพัฒนาซอร์สโค้ดลงเป็นอันมาก แต่ภาษาฮาสเกลมีลักษณะเป็น pure functional language ทำให้การเขียนฟังก์ชันไม่ยืดหยุ่นได้เท่าที่ควร ผู้วิจัยจึงปรับปรุงแนวทางการพัฒนาโปรแกรมเป็นการใช้ภาษาเออแลง ที่พบว่าช่วยให้การพัฒนาโปรแกรมทำได้สะดวกมากขึ้น เนื่องจากภาษาเออแลงมีลักษณะของภาษาเชิงประกาศ (declarative) ที่ช่วยให้การเขียนและอ่านโค้ดทำได้ง่ายกว่าภาษาฮาสเกล ซึ่งแสดงเปรียบเทียบรูปแบบการเขียนคำสั่งได้ดังรูปที่ 2.9

```
fac :: Int -> Int
fac 0 = 1
fac n = n * fac(n – 1)
```

(ก) ภาษาฮาสเกล

```
-module(fact).
-export([fac/1]).
fac(0) -> 1;
fac(N) when N > 0 -> N * fac(N – 1).
```

(ข) ภาษาเออแลง

รูปที่ 2.9 การเขียนฟังก์ชันแฟคทอเรียลเปรียบเทียบระหว่างภาษาฮาสเกลและภาษาเออแลง

โปรแกรมต้นแบบสำหรับงานการค้นพบรูปแบบที่ปรากฏบ่อยในภาษาเออแลง ที่แสดงด้านล่างนี้ เขียนขึ้นเพื่อค้นพบรูปแบบที่ปรากฏบ่อยในสายรหัสพันธุกรรม หรือดีเอ็นเอ โดยใช้ข้อมูล DNA-nominal จากฐานข้อมูล UCI (http://archive.ics.uci.edu/ml/datasets/) ไฟล์โปรแกรมจะเป็นชื่อเดียวกันกับชื่อโมดูลและมีส่วนขยายเป็น erl ดังนั้นโปรแกรมนี้จึงถูกบันทึกในชื่อ "assoDNA.erl" ข้อความที่ขึ้นต้นด้วยสัญลักษณ์ "%" เป็น comment

```
-module(assoDNA).
-import(lists,[seq/2,sum/1,flatten/1,split/2,nth/2,map/2,last/1]).
-import(io,[format/1,format/2]).
-import(ordsets,[to_list/1,from_list/1,is_subset/2,union/1]).
% Using this program:
%       c(assoDNA,[export_all]).
%        assoDNA:main2().
% input(_)->[[1,3,4],[2,3,5],[1,2,3,5],[2,5]].
% allItems() ->[1,2,3,4,5]. maxRec()->5. per()->50.
% -------------------------
% input(_)->[["a","c","d"],["b","c","e"],["a","b","c","e"],["b","e"]].
% allItems() ->["a","b","c","d","e"].
```

```
% maxRec()->5. per()->50.
%--------------------------
input(_) -> FileNs=["DNA-nominal.data","DNA-nominal.test"],
        format("~nFile 1.~p 2.~p ",FileNs),
        {_,FNo}=io:read(" :Choose> "),
        FileN=lists:nth(FNo,FileNs),
        format("Read from file:~p",[FileN]),
        readfile(FileN).

allItems() -> [F++[S]||F<- ["A","C","T","G"], S<-seq(048+1,048+60)].

readfile(FileName) ->  {ok, Binary} = file:read_file(FileName),
        Lines = string:tokens(erlang:binary_to_list(Binary), "\n\r"),
        L=lists:map(fun(X) -> string:tokens(X," ,") end,Lines),
        AD=[addCol(EachL,1) || EachL <-L],
        S=splitClass(["none","exon/intron","intron/exon"], AD),
        AllData=[extract(LL) || LL <-S],
        format("~nTher are 1-~w Classes",[length(AllData)]),
        {_,ClassNo}=io:read(" :Choose> "),
        {Class,Data}=lists:nth(ClassNo,AllData),
        io:format("Class =~p ",[Class]),
        Data.

splitClass([],_) -> [];
splitClass([H|T],L) -> [lists:filter(fun(X)->lists:last(X)==H end,L)|splitClass(T,L)] .

addCol([X],_) -> [X] ; % except the last
addCol([H|T],N) -> Col=048+N, [ H++[Col] | addCol(T,N+1)].

extract([H|T]) -> {last(H),extract2([H|T])}.
extract2(LL) -> [Rec--[last(Rec)]||Rec<-LL].

main() -> AllInput=input(1111),
        DB=myToSet(AllInput),Total=length(AllInput),
        {_,Per}=io:read(" input percent> "),
        MinSup=Total*Per/100,
        format("~nTotal=~w ,~w% MinSup=~w",[Total,Per,MinSup]),
        apriori(DB,allItems(),MinSup).
```

```
myToSet(L) -> [from_list(X)||X<-L].
myToList(SL) -> [to_list(S)||S<-SL].

apriori(DB, Items, Min) ->
        C1=[{from_list([X]),findSup(from_list([X]),DB)}|| X<-Items ],
        L1=[{FS,Sup} || {FS,Sup}<-C1,Sup>=Min],
        % print L1
        LkPrint=[ {to_list(FS),Sup,Sup/length(DB)*100} || {FS,Sup}<-L1],
        format("~nK=~w-~p,  has ~w set  ",[1,LkPrint,length(LkPrint)]),
        K=2,
        LS=[FS||{FS,_}<-L1],
        aprioriLoop(L1,DB,LS,K,Min) .

findSup(_, []) ->0;
findSup(Set, DB) -> [H|T]=DB,
        Cond = is_subset(Set,H),
        if Cond->1+findSup(Set,T);
            true -> findSup(Set,T)
        end.

aprioriLoop(AllL,_,[],_,_) -> AllL;
aprioriLoop(AllL,_,[_],_,_) -> AllL;
aprioriLoop(AllL,DB,LS,K,Min) -> Com=combi(LS),
        C_= myDistinct(usedCombi(Com,K)),
        Ck=[{X,findSup(X,DB)}|| X<-C_],
        Lk=[ {FS,Sup} || {FS,Sup}<-Ck,Sup>=Min],
        LkS=[FS||{FS,_}<-Lk],
        LkPrint=[ {to_list(FS),Sup,Sup/length(DB)*100} || {FS,Sup}<-Lk],
        format("~nK=~w-~p,  has ~w set  ",[K,LkPrint,length(LkPrint)]),
        aprioriLoop(AllL++Lk,DB,LkS,K+1,Min) .

myDistinct(List) -> to_list(from_list(List)).
combi([H|T]) -> [[H,Te] || Te<-T]++ combi(T);
combi([]) -> [].
usedCombi([H|T], K) -> Union=union(H),
        Len=ordsets:size(Union),
        if Len==K -> [Union|usedCombi(T,K)];
           true -> usedCombi(T,K)
        end ;
```

```
usedCombi([],_) -> [].
shift([H|T]) ->T++[H].

genR(_,Max,Max) -> [];
genR(L,N,Max) -> {H,T} = lists:split(N,L),
        [{H,T}]++genR(L,N+1,Max).

genRule(_,0,_) -> [];
genRule(L,Count,Len) -> genR(L,1,Len)++genRule(shift(L),Count-1,Len).

set(X) -> from_list(X).
list(X) -> to_list(X).

searchL(Set,[{Set,Val}|_])  ->  Val;
searchL(Set,[{_Another,_}|T])  -> searchL(Set,T);
searchL(_Set,[]) ->  1  .

findConf({H,B}, AllL)  -> {H,B,searchL(set(H++B),AllL)/searchL(set(H),AllL) }.

sortConf({_,_,Conf1},{_,_,Conf2}) -> Conf1>Conf2.

main2() ->
        format("~n----------START----------"),
        AllL=main(),
        AllAsso2=[list(X)|| {X,_} <-AllL,length(list(X))>1 ],
        AllRuleGen=lists:flatten([genRule(L,length(L),length(L))||L<-AllAsso2]),
        AllRuleConf=[findConf(X,AllL)||X<-AllRuleGen],
        format("~n~n AllRule=~w ,~nThere are ~w rules ",[AllRuleConf,length(AllRuleConf)]),
        lists:sort({assoDNA,sortConf},AllRuleConf),
        format("~n----------") .
```

```
% ---Test Module----------------------------
% c(assoDNA,[export_all]).
% assoDNA:findSupOf(["AM","GN"]).
% assoDNA:findPos(["AM","GN"]).
findPos([])->ok;
findPos([H|T])->[F,S]=H,format("~p~w--",[[F],S-48]),findPos(T).
subList(L1,L)->length(L--L1)==length(L)-length(L1).
findSup1(L1,LL)->lists:filter(fun(L)->subList(L1,L) end,LL) .
findSupOf(Lfind)-> format("\"DNA-nominal.data\",\"DNA-nominal.test\" "),
        {_,FileName}=io:read(" Start NewJob FileName> "),
        {ok, Binary} = file:read_file(FileName),
        Lines = string:tokens(erlang:binary_to_list(Binary), "\n\r"),
        L=lists:map(fun(X) -> string:tokens(X," ,") end,Lines),
        AD=[addCol(EachL,1) || EachL <-L],
        S1=splitClass(["none","exon/intron","intron/exon"],AD) ,
        [{_,CL1},{_,CL2},{_,CL3}] =[extract(LL1) || LL1 <-S1],
        OLen=[length(CL1),length(CL2),length(CL3)],
        Re=findSup1(Lfind,AD),
        S=splitClass(["none","exon/intron","intron/exon"],Re),
        AllData=[extract(LL) || LL <-S],
        myprint(OLen,AllData).

myprint([],_) -> endOfPrint;
myprint([OH|OT],[{C,LL}|T]) ->
        format("~nClass:~p has ~w = ~w percentOf ~w",[C,length(LL),length(LL)/OH*100,OH]),
        myprint(OT,T).
```