TRƯỜNG ĐẠI HỌC FPT

FPT UNIVERSITY

ASSIGNMENT 2
DATA STRUCTURE AND ALGORITHM

Người thực hiện:	**Nguyễn Đăng Lộc – SE160199,**
	Nguyễn Vĩ Khang – SE160221
Lớp:	**SE1616**
Khoá:	**16**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

TRƯỜNG ĐẠI HỌC FPT

FPT UNIVERSITY

ASSIGNMENT 2
DATA STRUCTURE AND ALGORITHM

Người thực hiện:	**Nguyễn Đăng Lộc – SE160199,**
	Nguyễn Vĩ Khang – SE160221
Lớp:	**SE1616**
Khoá:	**16**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

PHẦN ĐÁNH GIÁ CỦA GIẢNG VIÊN

_____

_____

_____

_____

_____

_____

_____

TP. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

MỤC LỤC

GIẢI THUẬT FP-GROWTH TÌM TẬP PHỔ BIẾN

Chương 1. GIỚI THIỆU

1.1. Tổng quan

Nhờ những thành tựu lớn lao trong nghiên cứu khoa học, sự phát triển của công nghệ thông tin ngày càng mang lại nhiều hiệu quả thiết thực đối với các công việc trong cuộc sống con người. Các phương tiện lưu trữ ngày càng nhỏ gọn nhưng khả năng lưu trữ thì lại càng lớn, các hệ quản trị cơ sở dữ liệu cung cấp cho người dùng khả năng lưu trữ không giới hạn. Và người dùng, ngoài nhu cầu lưu trữ còn mong muốn hiểu được những tri thức, thông tin tiềm ẩn trong những kho dữ liệu khổng lồ ấy. Quá trình hiểu dữ liệu, tìm kiếm ý nghĩa của những thông tin đã lưu trữ được gọi là khai phá dữ liệu. Theo chiều dài thời gian, các phương pháp đã được đưa ra và cải thiện để có thể khai thác những giá trị tri thức từ dữ liệu, vận dụng giải quyết các bài toán quan trọng trong nhiều lĩnh vực của cuộc sống.

Quá trình khai phá dữ liệu là quá trình trích xuất thông tin, khám phá tri thức có mối tương quan nhất định từ một kho dữ liệu khổng lồ nhằm mục đích dự đoán các xu thế, hành vi trong tương lai, hoặc tìm kiếm tập các thông tin hữu ích mà bình thường không thể nhận diện được.

Một trong những nội dung cơ bản nhất và cũng rất quan trọng trong khai phá dữ liệu là tìm các mẫu phổ biến. Phương pháp này nhằm tìm ra các tập phần tử thường xuất hiện đồng thời trong cơ sở dữ liệu và rút ra các luật về ảnh hưởng của một tập phần tử dẫn đến sự xuất hiện của một (hoặc một tập) phần tử khác như thế nào.

Tập phổ biến *(Frequent patterns)* là những mẫu (tập món đồ, danh sách con, cấu trúc con) mà thường xuyên xuất hiện trong một tập dữ liệu. Ví dụ, tập các món hàng như sữa và bánh mì, 2 món hàng thường xuyên xuất hiện cùng nhau trong 1 giao dịch thì được gọi là 1 tập phổ biến. Một dãy các hành động, như là đầu tiên mua máy tính, sau đó mua máy ảnh, và mua thẻ nhớ, nếu chúng thường xuyên xuất hiện trong lịch sử mua hàng của CSDL thì được gọi là một tập phổ biến.

Tìm ra các tập phổ biến đóng một vai trò quan trọng trong khai phá luật kết hợp *(associations),* phân tích tương quan *(correlations),* và rất nhiều mối quan hệ thú vị khác trong kho lưu trữ dữ liệu. Ngoài ra, nó còn giúp phân loại, gom nhóm dữ liệu và thực thi nhiều tác vụ khai phá dữ liệu khác. Do đó, khai phá tập phổ biến *(frequent pattern mining)* đã trở thành một nhiệm vụ và chủ đề được chú trọng khi nghiên cứu dữ liệu. [1]

1.2. Bài toán cụ thể đặt ra

Cho tập các món hàng $I = \{i_1, i_2, \dots, i_m\}$ và cơ sở dữ liệu D là các giao dịch, mỗi giao dịch là một tập con của tập các món hàng I ($T \subseteq I$).

Tìm tất cả các tập con của I là tập món hàng thường xuyên xuất hiện cùng nhau trong các giao dịch.

1.3. Ý nghĩa của việc giải quyết bài toán

Khai phá tập phổ biến thực hiện tìm kiếm, nghiên cứu các mối quan hệ trong một hệ CSDL. Khai phá các tập món hàng dẫn đến một khám phá thú vị về sự kết hợp, tương hợp giữa các món hàng trong những bộ dữ liệu khổng lồ. Với số lượng lớn dữ liệu liên tục được thu thập và lưu trữ, nhiều công việc kinh doanh, các ngành công nghiệp ngày càng muốn khai phá những mẫu từ CSDL. Một trong những ứng dụng cụ thể nhất của công việc khai phá tập phổ biến đó là Phân tích giỏ thị trường *(Market Basket Analysis)*

Market Basket Analysis

Quy trình này khai thác tập dữ liệu lớn như lịch sử mua hàng, phân tích thói quen mua hàng của khách bằng các tìm ra các tập món hàng thường xuyên được mua, sự kết hợp giữa các món hàng khác nhau mà người dùng cho vào giỏ hàng của họ. Sự khám phá đó có thể hỗ trợ người bán hàng trong các chiến dịch marketing bằng cách tìm sự thật ngầm hiểu *(insight)*. Các ứng dụng của MBA:

- **Vị trí sản phẩm**: Xác định các món hàng thường có thể được mua cùng nhau và sắp xếp vị trí của các mặt hàng đó (trên catalog, websites,..) gần nhau để kích thích người mua.
- **Sắp xếp giá bán hàng:** Một cách khác với cách trên là tách các mặt hàng thường được mua cùng, khuyến khích khách hàng đi quanh trong cửa hàng để tìm thứ họ đang tìm để có khả năng mua, tăng xác suất mua hàng bổ sung.
- **Bán gia tăng, bán chéo và bán nhóm hàng:** Các công ty có thể sử dụng nhóm sở thích của nhiều sản phẩm như một dấu hiệu cho thấy khách hàng có thể có xu hướng mua các sản phẩm được nhóm cùng một lúc. Điều này cho phép trình bày các mặt hàng để bán kèm hoặc có thể gợi ý rằng khách hàng có thể sẵn sàng mua nhiều mặt hàng hơn khi một số sản phẩm nhất định được đóng gói cùng nhau.
- **Duy trì khách hàng:** Có thể dùng phân tích giỏ thị trường để đưa ra khuyến mãi phù hợp, cung cấp cho khách hàng, duy trì hoạt động mua hàng của khách. [2]

Ngoài ra, việc giải quyết bài toán tìm tập phổ biến nói chung còn có ý nghĩa thực tiễn trong các tác vụ, công việc thuộc các lĩnh vực khác như: chiến lược định giá lỗ, chiến lược giảm giá, hệ thống đề xuất, phân tích phân nhóm các tài liệu, phân tích nhật ký web, phân tích trình tự DNA,… và cùng với rất nhiều ứng dụng trong các lĩnh vực: y học, tài chính, bảo hiểm, khí tượng, tin – sinh,…

1.4. Mục tiêu

Trong bài báo cáo này, chúng tôi sẽ nói về khái niệm tập phổ biến, giải thuật FP-Growth, và trình bày làm sao có thể ứng dụng giải thuật FP-Growth để thực hiện khai phá luật phổ biến các hóa đơn bán hàng.

Chủ đề khai phá tập phổ biến thực sự là rất rộng lớn. Trong quy mô đề tài, chúng tôi tập trung bàn về một phương pháp cụ thể của khai phá tập phổ biến ứng dụng thuật toán FP-Growth. Chúng tôi sẽ trả lời cho các câu hỏi: Tập phổ biến là gì?

Nó có ý nghĩa, vai trò như thế nào? Giải thuật FP-Growth thực thi như thế nào? Ứng dụng giải thuật FP-Growth để giải quyết bài toán tìm tập phổ biến các hóa đơn bán hàng? Tính hiệu quả của giải thuật FP-Growth và các biện pháp cải thiện nếu có?

Với những gì đã đề cập ở trên, đề tài sẽ tập trung vào những vấn đề sau:

- Thực hiện khảo sát, phân tích và triển khai giải thuật FP-Growth bằng ngôn ngữ Java.
- Khảo sát tập dữ liệu hóa đơn bán hàng online của UK, France, Portual, Sweden, tìm các tập món hàng phổ biến bằng giải thuật FP-Growth đã triển khai.
- Thực hiện so sánh kết quả của chương trình tự triển khai với kết quả từ công cụ Weka.
- Đánh giá kết quả đã đạt được.

1.5. Phạm vi

Đề tài sử dụng dữ liệu hóa đơn bán lẻ của một cửa hàng online đối với các nước France, United Kingdom, Portugal, Sweden từ 01/12/2010 - 09/12/2011. *(Truy cập tại: http://archive.ics.uci.edu/ml/datasets/Online+Retail)*

Giải thuật sử dụng là FP-Growth tự triển khai, thực hiện tìm các tập hạng mục phổ biến.

Những vấn đề nâng cao hơn có liên quan như *Closed Itemsets, Association Rules, Correlations,…* sẽ không nằm trong phạm vi của báo cáo bài tập này.

Chương 2. GIẢI QUYẾT BÀI TOÁN

Để giải quyết bài toán đã đặt ra, bài toán cần được mô hình hóa, biểu diễni qua những định nghĩa, khái niệm dưới.

2.1. Các khái niệm, lý thuyết cơ bản

a. Hạng mục *(item)* và tập hạng mục *(itemset)*

Cho tập gồm *N* đối tượng $I = \{I_1, I_2, I_3, \dots, I_n\}$, mỗi phần tử $I_i \in$ I được gọi là một hạng mục *(item)*.

Một tập con bất kỳ $X \subseteq I$ được gọi là một tập hạng mục *(itemset)*.

X gọi là một tập hạng mục mức k *(k-itemset)* nếu X chứa k hạng mục.

Ví dụ: $I = \{1, 2, 3, 4, 5\}$

1, 2, 3, 4, 5 là các hạng mục.

X = {2 5} là một tập hạng mục mức 2 *(2-itemset)*.

b. Giao dịch *(transaction)*

Cho tập $D = \{T_1, T_2, T_3, \dots, T_m\}$, mỗi phần tử $T_j \in D$ được gọi là một giao dịch *(transaction)* và là một tập con nào đó của I ($T_j \subseteq I$).

Người ta gọi D là cơ sở dữ liệu giao dịch *(transaction database)*. Số giao dịch có trong D ký hiệu là $|D|$.

Ví dụ: $I = \{A, B, C, D, E, F\}$. Cơ sở dữ liệu giao dịch D gồm các tập con T_j khác nhau của I:

$T_1 = \{A, B, C, D\}$

$T_2 = \{A, C, E\}$

$T_3 = \{A, E\}$

$T_4 = \{A, E, F\}$

$T_5 = \{A, B, C, E, F\}$

c. Độ hỗ trợ *(support)* với một tập hạng mục

Độ hỗ trợ ứng với tập hạng mục X là xác suất xuất hiện của X trong cơ sở dữ liệu giao dịch D (tỷ lệ các giao dịch có chứa X trên tổng số các giao dịch có trong cơ sở dữ liệu giao dịch D)

Công thức: $sup(X) = \frac{|\{T \in D \mid X \subseteq T\}|}{|D|} = \frac{C(X)}{|D|}$, $C(X)$ là số lần xuất hiện của tập X

Ví dụ: Theo ví dụ trên, có: $|D| = 5, X = \{A, E\} \rightarrow C(X) = 4$

$sup(X) = \frac{C(X)}{|D|} = \frac{4}{5} = 0.8$ (hay 80%).

d. Tập phổ biến/ tập thường xuyên *(Large itemset/frequent itemset)*

Các tập hạng mục có độ hỗ trợ lớn hơn một giá trị ngưỡng *minSup* nào đó cho trước được gọi là các tập phổ biến *(frequent item set)*

Một tập hạng mục X được gọi là phổ biến hay tập thường xuyên nếu:

$$sup(X) \geq \delta$$

δ: minSupp là một ngưỡng do người dùng xác định.

Ngược lại, X là tập không phổ biến *(small itemset).*

e. Cơ sở lý thuyết

Nguyên lý Apriori ***"Nếu một tập hạng mục là tập phổ biến thì mọi tập con khác rỗng bất kỳ của nó cũng là tập phổ biến".*** [3]

Chứng minh:

Xét $X' \subseteq X$. δ là ngưỡng độ hỗ trợ $minSup$. Một tập hạng mục xuất hiện bao nhiêu lần thì các tập con chứa trong nó cũng xuất hiện ít nhất bấy nhiêu lần, nên ta có:

$$C(X') \geq C(X) \qquad (1)$$

X là tập phổ biến nên:

$$sup(X) = \frac{C(X)}{|D|} \geq \delta \Rightarrow \mathrm{C(X)} \geq \delta|\mathrm{D}| \qquad (2)$$

Từ (1) và (2) suy ra:

$$C(X') \geq \delta|\mathrm{D}| \Rightarrow sup(X') = \frac{C(X')}{|D|} \geq \delta$$

Do đó, X' cũng là tập phổ biến.

2.2. Mô tả cấu trúc dữ liệu

Để triển khai giải thuật FP-Growth, cần sử dụng cấu trúc dữ liệu FP-tree. FP-tree là mở rộng của cấu trúc cây tiền tố *(Prefix Tree)*, được gọi là cây mẫu phổ biến *(Frequent Pattern Tree*, gọi tắt *FP Tree*) dùng để nén dữ liệu.

a. Cây tiền tố

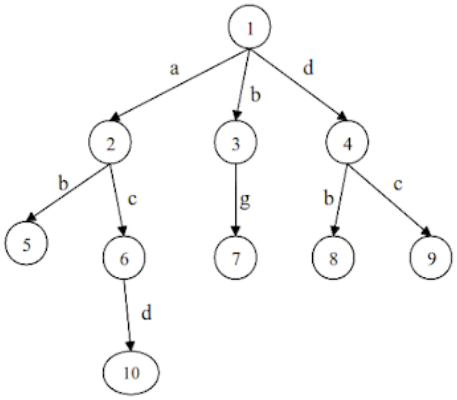
Cây tiền tố (*Prefix tree*, hay còn gọi là *Trie*) là một cấu trúc dữ liệu dùng để quản lý một tập hợp các xâu. *Prefix tree* cho phép:

- Thêm một xâu vào tập hợp
- Xóa một xâu vào tập hợp
- Kiểm tra một xâu có tồn tại trong tập hợp hay không.

Cấu trúc của cây tiền tố

Cây tiền tố gồm một gốc không chứa thông tin (NULL), trên mỗi cạnh lưu một ký tự, mỗi nút và đường đi từ gốc đến nút đó thể hiện 1 xâu, gồm các ký tự là các ký tự thuộc cạnh trên đường đi đó. [4]

Ví dụ: Nút 1 là nút gốc, nút 6 thể hiện có 1 xâu là ‘ac’, nút 9 thể hiện có 1 xâu là ‘dc’, nút 10 thể hiện có 1 xâu là ‘acd’.



Hình 2-1. Ví dụ cây tiền tố

Tùy theo yêu cầu của bài toán cần xử lý mà mỗi nút sẽ có thể cần lưu thêm các thông tin khác nhau.

Các thao tác trên cây tiền tố

Tạo nút

```
static class PTNode {
  PTNode [] child = new PTNode[ALPHABET_SIZE];
  boolean isEndOfWord;
  PTNode(){
     isEndOfWord = false;
     for (int i = 0; i < ALPHABET_SIZE; i++)
     child[i] = null;
  }
};
```

Chèn xâu vào cây tiền tố

```
static void insert(String key) {
  PTNode curNode = root;
  for (int level = 0; level < key.length(); level++) {
     int index = key.charAt(level) - 'a';
     if (curNode.children[index] == null)
        curNode.child[index] = new TrieNode();
     curNode = curNode.child[index];
  }
  curNode.isEndOfWord = true;
};
```

b. Cây mẫu phổ biến

Cây mẫu phổ biến *(FP-tree)* được xây dựng, để nén dữ liệu các giao dịch, sau đó thuật toán FP-Growth sẽ thực hiện khai phá FP-tree.

Cấu trúc của cây mẫu phổ biến

- Gốc của cây *(root)* gán nhãn NULL, mỗi đường đi trên cây biểu diễn một tập hạng mục *(itemset)*.
- Mỗi nút *(node)*, trừ gốc, gồm 5 thông tin, bao gồm: item-name *(key)*, số đếm *(supportCount)*, nút cha *(parentNode)*, nút kế tiếp có cùng item-name trong cây *(nextNode)*, và một mảng chứa các nút con *(childs[])*.

Các thao tác trên cây mẫu phổ biến

- Thêm nút: Thủ tục thêm nút:

```
Procedure insert(itemList p|P, Node T):
    Gọi N là nút con của T có item-name là p:
      + Nếu N tồn tại, tăng số đếm (count) của N lên 1.
      + Nếu không, tạo N và đặt các giá trị của nút (*),
        thêm N vào danh sách nút con của T.
    Nếu P còn hạng mục, tiếp tục đệ quy hàm insert(P, N)
    Nếu không, quay về
```

- Duyệt từ 1 nút về gốc (*reverse traverse)*

```
Procedure reverse(FPNode node):
  Nếu curNode != null:
    Truy ngược về nút cha của curNode
```

Ngoài ra, ta còn sử dụng thêm bảng liên kết *(Header Table)* để kết nối các nút có cùng *item-name* trong FP-Tree.

Tạo bảng liên kết gồm 2 thông tin: key và nút mới nhất có *item-name = key*.

```
Ở (*):
 + Nếu bảng rỗng, ta sẽ gán headerTable[key] = nút mới
 + Ngược lại, ta gán nextNode của nút mới là headerTable[key],
   gán headerTable[key] là nút mới.
    '
```

Đặc điểm của FP-tree phù hợp cho khai thác mẫu phổ biến:

- Liên kết nút *(Node Links)*: Đối với mỗi *item i* phổ biến, tất cả các mẫu phổ biến có thể có chứa *item i* đều có thể thu được bằng cách đi theo các liên kết nút của *i*, bắt đầu từ *head* của *i* trong Header Table.
- Đường tiền tố *(Prefix Path)*: Để tính các mẫu phổ biến của nút *item i* trong đường dẫn P, chỉ cần gom tích lũy đường dẫn tiền tố con của *item i* trong P và *support* của nó bằng với *support* của nút *item i*.

2.3. Sơ đồ giải thuật

a. Ý tưởng giải thuật

Nén cơ sở dữ liệu các hạng mục trong từng giao dịch vào cấu trúc cây mẫu phổ biến *(Frequent Pattern Tree*, gọi tắt *FP Tree*).

Thực hiện khai phá phát triển *(growth)* từng đoạn dựa trên FP Tree.

Dựa vào phương pháp chia để trị *(divide-and-conquer)* để phân rã *(decompose)* nhiệm vụ khai phá thành tập các nhiệm vụ nhỏ hơn với giới hạn các hạng mục mẫu.

Phương án này giúp thu gọn không gian tìm kiếm, chỉ kiểm tra với các tập dữ liệu con, cho phép phát hiện ra các tập phổ biến mà không cần tạo ra các ứng viên.

b. Các bước giải thuật

```
Input:
  D – cơ sở dữ liệu các giao dịch
  minSupp – giá trị độ hỗ trợ nhỏ nhất
Output: danh sách các tập phổ biến
```

<u>Bước 1:</u> Xây dựng FP Tree

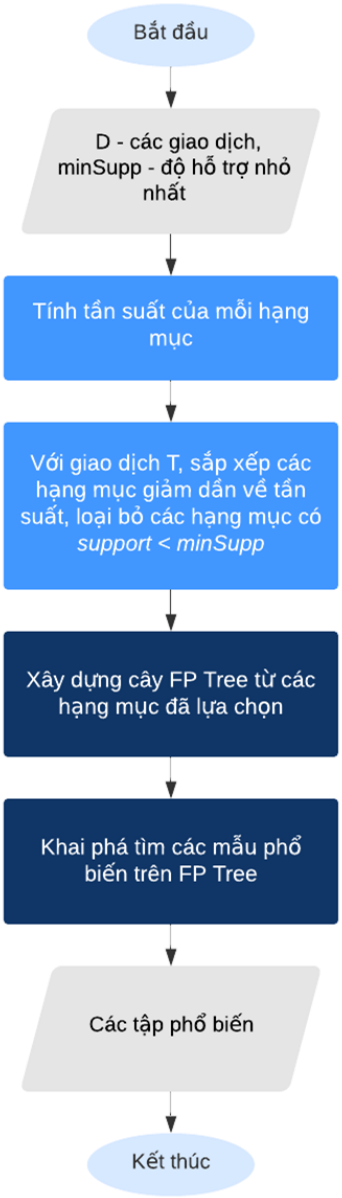
(1) Duyệt CSDL lần 1, tính tần suất của mỗi hạng mục (đếm số lần xuất hiện)

(2) Duyệt CSDL lần 2, với mỗi giao dịch T, sắp xếp các hạng mục giảm dần về tần suất, loại bỏ các hạng mục có độ hỗ trợ không thỏa mãn *(support < minSupp)*, được danh sách các hạng mục của T đã sắp xếp.

(3) Các hạng mục ở trên được đưa vào FP Tree.

<u>Bước 2:</u> Thực hiện thuật toán FP-Growth

(4) Khai phá tìm các mẫu phổ biến trên FP Tree đã xây mà không cần duyệt lại CSDL.



Hình 2-2. Sơ đồ giải thuật tổng quát

Giải thuật xây dựng FP Tree

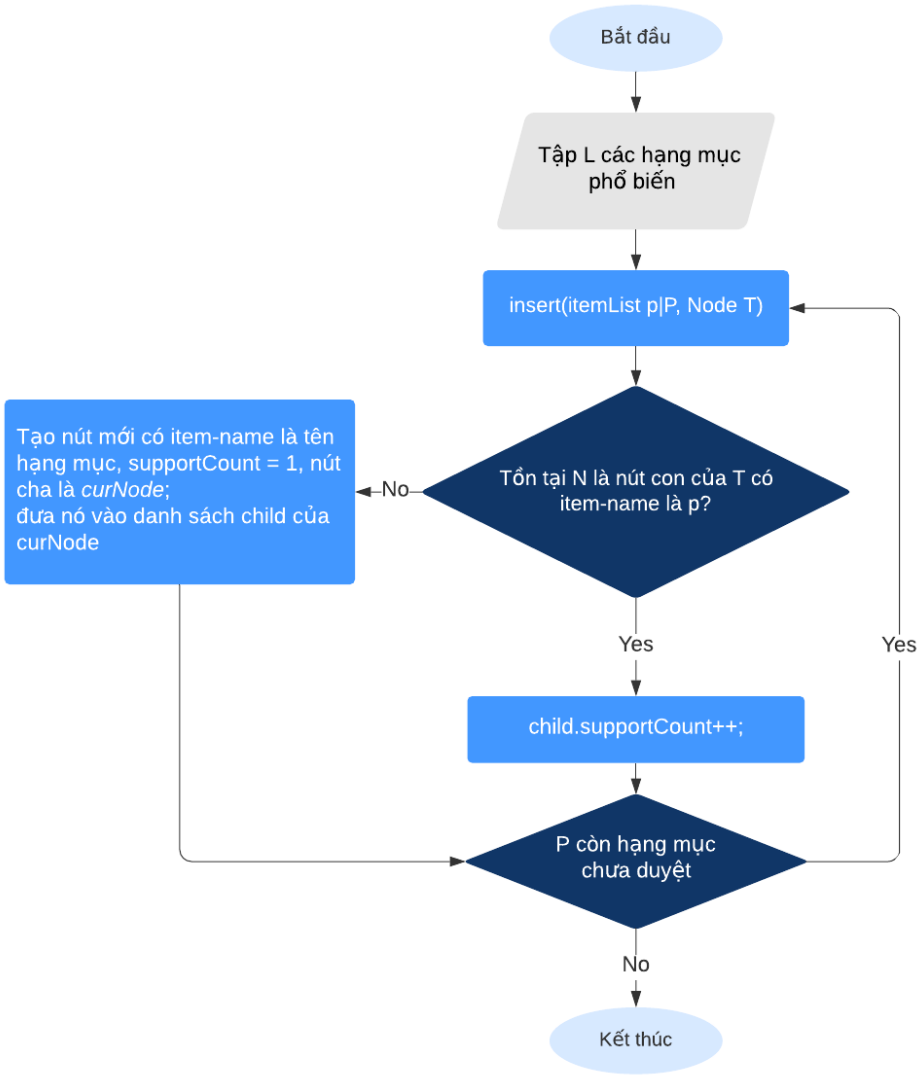
Sau khi thực hiện xong thao tác (2) trên, được danh sách *L* (hạng mục đã chọn lọc sắp xếp) của mỗi giao dịch. Tiếp theo, đưa các hạng mục trong *L* vào FP-tree.

+ Thứ tự sắp xếp của các mục được tuân thủ trong suốt quá trình xây dựng cây FP.
+ Các đường đi có thể có thể có những đoạn trùng nhau do các giao dịch có các hạng mục chung (chung tiền tố trong dãy). Mỗi lần có phần tử trùng thì trọng số của đỉnh ở vị trí trùng được tăng lên 1.

Tạo một gốc *(root)* cho FP-tree, và gán nhãn *null*.

Gọi tập hạng mục phổ biến được sắp xếp *L* là *[p|P]*, với p là phần tử đầu tiên trong P, và P là phần còn lại của danh sách. *T* là *root* của FP-tree. Gọi hàm insert_tree([p|P], T) để chèn L vào FP-tree. [2]

```
Procedure insert(itemList p|P, Node T):
   Gọi N là nút con của T có item-name là p:
    + Nếu N tồn tại, tăng số đếm (count) của N lên 1.
    + Nếu không, tạo nút mới N (count=1, N.parent=T),
      thêm N vào danh sách nút con của T,
      N.nextNode sẽ liên kết tới node cùng item-name qua headerTable.
   Nếu P còn hạng mục, tiếp tục đệ quy hàm insert(P, N).
   Nếu không, quay về.
```



Hình 2-3. Sơ đồ giải thuật xây dựng FP-tree

Đặc điểm phát triển mẫu *(pattern growth)*

Gọi α là một tập phổ biến của cơ sở dữ liệu, M là cơ sở mẫu điều kiện của α, β là một tập hạng mục trong M. Khi đó $\alpha \cup \beta$ là một tập phổ biến trong cơ sở dữ liệu khi và chỉ khi β là phổ biến trong M.

Ví dụ: Xét *"fcabm"* là đường đi trên FP-tree:

+ *"fcab"* là cơ sở mẫu điều kiện của *"m"*
+ *"b"* không phổ biến trong tập các mẫu chứa "fcab"

→ *"bm"* không là một mẫu phổ biến.

Giải thuật FP-Growth

(4.1) Duyệt Header Table theo thứ tự tăng dần độ hỗ trợ. Với mỗi hạng mục *i*, tạo cơ sở mẫu điều kiện *(CPB - conditional pattern base)*

- Mỗi nút của FP-tree tương ứng với một hạng mục, được gắn trọng số là số lần xuất hiện.
- Mỗi mẫu có điều kiện là một đường đi từ gốc tới cha của đỉnh có chứa *i (prefix-path)*. Mỗi mẫu được gán trọng số bằng với trọng số của đỉnh chứa *i* ở cuối đường đi.
- Duyệt cây từ nút về gốc, ta có một *prefix-path*, sau đó dùng liên kết nút (*Node Links*) để chuyển sang các nút có cùng *item-name* trong FP-tree.
- Tổng hợp tất cả các *prefix-path* của hạng mục đang xét để tạo thành cơ sở mẫu điều kiện *(CPB)*.

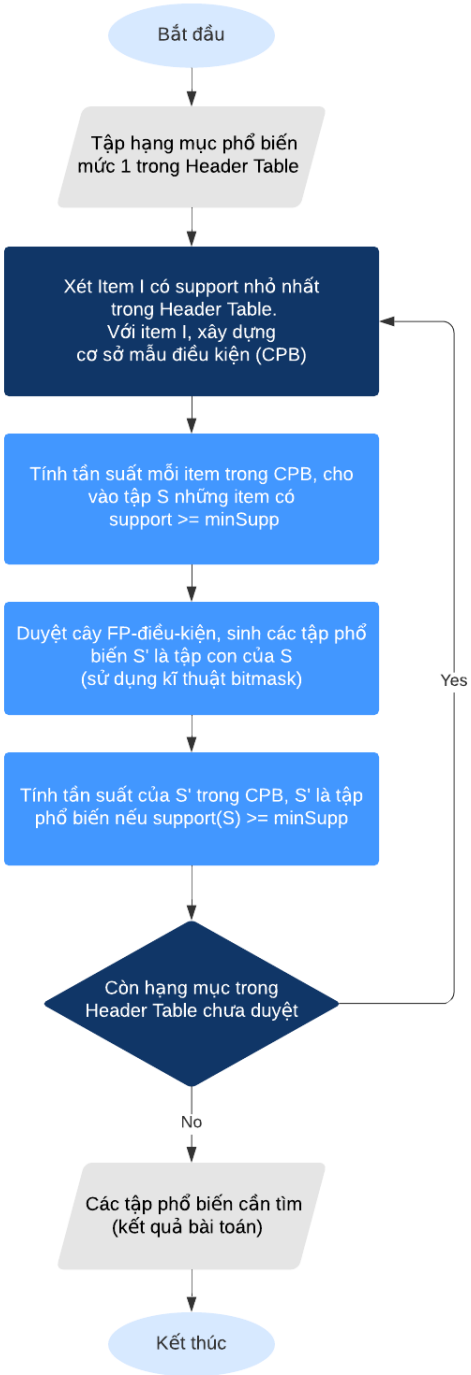
(4.2) *countFreq[j]* là tần suất của hạng mục *j* trong *CPB*. Tạo tập *S* gồm hạng mục có *countFreq[j]* ≥ *minSupp*.

Xây dựng cây FP-điều-kiện từ tập S

(4.3) Duyệt cây FP-điều-kiện để sinh các tập phổ biến có hậu tố là *i*.

Sử dụng phương pháp trạng thái bit *(bitmask)* để duyệt từng tập con *S'* của *S*, với điều kiện là *S'* chứa *i*.

(4.4) Tính tần suất của *S'* trong cơ sở mẫu điều kiện. *S'* là một tập phổ biến khi *freq(S')* ≥ *minSupp*.



Hình 2-4. Sơ đồ giải thuật FP-Growth

Ví dụ:

Cơ sở dữ liệu giao dịch D như bảng dưới, minSupp = 3.

TID	Tập hạng mục
1	B, A, T
2	A, C
3	A, S
4	B, A, C
5	C, B, S
6	C, A, S
7	B, S
8	B, A, S, T
9	B, A, S

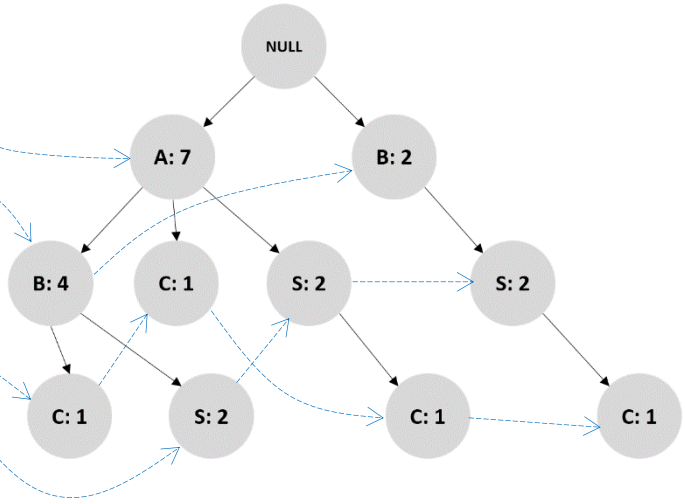
Thao tác (1), (2)

T có support = 2 (loại)

TID	Tập hạng mục	Hạng mục phổ biến
1	B, A, T	A, B
2	A, C	A, C
3	A, S	A, S
4	B, A, C	A, B, C
5	C, B, S	B, S, C
6	C, A, S	A, S, C
7	B, S	B, S
8	B, A, S, T	A, B, S
9	B, A, S	A, B, S

Tạo Header Table, xây dựng FP-tree – thao tác (3)

Hạng mục	Tần suất	Liên kết nút
A	7	
B	6	
S	6	
C	4	



Thực hiện FP-Growth, khai phá dữ liệu trên FP-tree – thao tác (4)

Hạng mục	Cơ sở mẫu điều kiện	FP-tree điều kiện	Mẫu phổ biến
C	{AB: 1, A: 1, AS: 1, BS: 1}	{A: 3} - C	{AC: 3}
S	{AB: 2, A: 2, B: 2}	{A: 4, B: 4} – S	{AS: 4, BS: 4}
B	{A: 4}	{A: 4} – B	{A,B:4}
A	∅	∅ - A	{A: 7}

2.4. Hiện thực

a. Tạo Node

```
public class FPNode {
  String itemID = null;
  int supportCnt = 0;
  HashMap<String, FPNode> child = new HashMap<>();
  FPNode pre = null;
  FPNode nxt = null;


  public FPNode() {}
  public FPNode(String itemID) {
      this.itemID = itemID;
  }
}
```

b. Tạo FP-tree

```
public class FPTree {
  FPNode root;
  int supportCnt = 0

  public FPTree() {
     root = new FPNode();
  }

  private FPNode insert(FPNode cur, ArrayList<String> arr, int id,
                    HashMap<String, LinkedList<FPNode> > headerTable) {
     if(arr.size() == id) return cur;
     String product = arr.get(id);
     if(cur.child.get(product) != null)
        cur.child.get(product).supportCnt++;
     else {
        cur.child.put(product, new FPNode(product));
        cur.child.get(product).supportCnt = 1;
        cur.child.get(product).pre = cur;
        headerTable.get(product).add(cur.child.get(product));
     }
     if(id + 1 < arr.size()
        cur.child.put(product, insert(cur.child.get(product), arr,
                       id + 1, headerTable));
     return cur;
  }

  public void insert(ArrayList<String> transaction, HashMap<String,
                     LinkedList<FPNode> > headerTable) {
     root = insert(root, transaction, 0, headerTable);
  }

  public ArrayList<FPNode> reverse(FPNode node) {
     ArrayList<FPNode> result = new ArrayList<>();
     while(node != null) {
        result.add(node);
        node = node.pre;
     }
     return result;
  }
}
```

c. Thực hiện giải thuật FP-Growth

Khai báo

```
int cntTransaction;
int relativeMinsupp;
HashMap<String, LinkedList<FPNode> > headerTable = new HashMap<>();

ArrayList<ArrayList<String>> itemsets = new ArrayList<>();
// lưu CSDL các hạng mục của từng giao dịch

HashMap<String, Integer> itemSupport = new HashMap<>();
// độ hỗ trợ (~ tần suất) của từng hạng mục

ArrayList<ArrayList<String>> itemPrefixPaths = new ArrayList<>();
// cơ sở mẫu điều kiện với hạng mục i

ArrayList<Integer> prefixPathSupp = new ArrayList<>();
// độ hỗ trợ của mỗi hạng mục trong mẫu điều kiện

ArrayList<ArrayList<String> > freqItemset = new ArrayList<>();
// lưu danh sách tập phổ biến cần tìm

ArrayList<Integer> freqResult = new ArrayList<>();
// độ hỗ trợ của tập phổ biến
```

Đọc dữ liệu

```
private void convert(String data) {
  String[] list = data.split(" ");
  ArrayList<String> items = new ArrayList<>();

  for(String item : list)
      items.add(item.trim());

  itemsets.add(items);
  ++cntTransaction;
}

private void readData() {
  File f = new File(input);
  try {
      Scanner sc = new Scanner(f);

      while(sc.hasNextLine()) {
          String data = sc.nextLine();
          convert(data);
      }
  } catch (FileNotFoundException ex) {
      System.out.println("File not found!");
  }
}
```

(1) Duyệt CSDL lần 1, tính tần suất của mỗi hạng mục

```
// (1) First scan: calculate frequency of item
private void calcFrequency() {
  for(ArrayList<String> itemset : itemsets) {
      for(String item : itemset) {
          itemSupport.merge(item, 1, (a, b) -> a + b);
      }
   }
}
```

(2) Duyệt CSDL lần 2, sắp xếp các hạng mục giảm dần về tần suất, loại bỏ các hạng mục có độ hỗ trợ không thỏa mãn.

```
// (2) Second scan: build list L
public void prepareData() {
  for(ArrayList<String> itemset : itemsets) {
      Collections.sort(itemset, (var o1, var o2) -> {
          int cmp = itemSupport.get(o2) - itemSupport.get(o1);
          return (cmp == 0 ? o1.compareToIgnoreCase(o2) : cmp);
      });

      while(!itemset.isEmpty() &&
itemSupport.get(itemset.get(itemset.size() - 1)) < relativeMinsupp) {
          itemset.remove(itemset.size() - 1);
      }

      for(String item : itemset) {
          if(headerTable.get(item) == null)
              headerTable.put(item, new LinkedList<>());
      }
   }
}
```

(3) Các hạng mục ở trên được đưa vào FP Tree.

```
// (3) Recursively add item in L to FP-tree
public void buildFPTree() {
  for(int i = 0;i < cntTransaction;++i) {
      myFPTree.insert(itemsets.get(i), headerTable);
  }
}
```

(4.1) Duyệt Header Table theo thứ tự tăng dần độ hỗ trợ. Với mỗi hạng mục *i*, tạo cơ sở mẫu điều kiện *(CPB - conditional pattern base)*

```
public void runFP-Growth() {
  ArrayList<String> freqItem = new ArrayList<>(headerTable.keySet());

  Collections.sort(freqItem, (String o1, String o2) -> {
      int cmp = itemSupport.get(o2) - itemSupport.get(o1);
      return (cmp == 0 ? o1.compareToIgnoreCase(o2) : cmp);
  });

  // (4.1) Duyệt Header Table theo thứ tự tăng dần độ hỗ trợ
  //       tạo cơ sở mẫu điều kiện
  for(int i = freqItem.size() - 1; i >= 0; --i) {
      reset();

      String item = freqItem.get(i);

      // Danh sách các nút cùng thể hiện hạng mục i trên FP-tree
      LinkedList<FPNode> nodeLinks = headerTable.get(item);
      for(FPNode node : nodeLinks) {
          ArrayList<FPNode> prefixPath = myFPTree.reverse(node);
          addToContainer(prefixPath);
      }

      mineData(item);
  }
}

// Tổng hợp các prefix-path của hạng mục i đang xét
// để tạo thành cơ sở mẫu điều kiện
void addToContainer(ArrayList<FPNode> prefixPath) {
  ArrayList<String> pattern = new ArrayList<>();

  for(FPNode node : prefixPath)
      if(node.item != null)
          pattern.add(node.item);

  Collections.reverse(pattern);

  // thêm 1 tập hạng mục vào CPB
  itemPrefixPaths.add(pattern);

  // support của các hạng mục trong tập = support của hạng mục i
  prefixPathSupp.add(prefixPath.get(0).supportCnt);
}

public void reset() {
  itemPrefixPaths.clear();
  prefixPathSupp.clear();
}
```

(4.2) Tạo tập *S*: *countFreq[j]* ≥ *minSupp*. Xây dựng cây FP-điều-kiện từ tập S.

(4.3) Duyệt cây FP-điều-kiện để sinh các tập phổ biến có hậu tố là *i*.

(4.4) *S'* là một tập phổ biến khi *freq(S')* ≥ *minSupp*.

```
// lưu kết quả
void getFreqPattern(Set<String> list, String str, int freqCount) {
  Iterator<String> ptr = list.iterator();
  String tmp = null;
  ArrayList<String> ans = new ArrayList<>();
  while(ptr.hasNext()) {
      tmp = ptr.next();
      if(tmp == str) continue;
      ans.add(tmp);
  }
  ans.add(str);
  freqItemset.add(ans);  // một mẫu phổ biến
  freqResult.add(freqCount);  // support của mẫu phổ biến tương ứng
}

void mineData(String curItem) {
  HashMap<String, Integer> countFreq = new HashMap<>();

  // tính tần suất (~support) của các hạng mục trong PCB
  for(int i = 0;i < itemPrefixPaths.size();++i){
      ArrayList<String> prefixPath = itemPrefixPaths.get(i);
      // support của các hạng mục trong tập = support của hậu tố i
      for(String item : prefixPath)
          countFreq.merge(item, prefixPathSupp.get(i), (a, b)-> a + b);
  }

  // (4.2) Tạo tập S - các frequent itemset tạo từ CPB
  ArrayList<String> valid = new ArrayList<>();
  for(Map.Entry<String, Integer> item : countFreq.entrySet()) {
      if(item.getValue() >= relativeMinsupp)
          valid.add(item.getKey());
  }

  // (4.3) Dùng kĩ thuật trạng thái bit (bitmask) duyệt tập con của S
  for(int state = 1; state <= Math.pow(2, valid.size()) - 1; ++state) {
      Set<String> comb = new HashSet<>();
      for(int j = 0;j <= sz - 1;++j)
          if((state >> j & 1) == 1) {
              comb.add(valid.get(j));
          }
      if(!comb.contains(curItem)) continue;

      // (4.4) Tính tần suất của tập S'
      int freqCount = 0;
      for(int i = 0;i < itemPrefixPaths.size();++i) {
          ArrayList<String> prefixPath = itemPrefixPaths.get(i);
          HashSet<String> prefix = new HashSet<>(prefixPath);
          prefix.retainAll(comb);
          if(prefix.size() == comb.size())
              freqCount += prefixPathSupp.get(i);
      }
      if(freqCount >= relativeMinsupp)
          getFreqPattern(comb, curItem, freqCount);
  }
}
```

In kết quả

```
private void printFile() {
  try {
    PrintWriter fw = new PrintWriter(output);
    int it = 0;

    StringBuilder sb = new StringBuilder("");
    for(ArrayList<String> itemset : freqItemset) {
      for (int i = 0; i < itemset.size(); i++) {
        sb.append(itemset.get(i));
        if (i != itemset.size() - 1) sb.append(" ");
      }
      sb.append(":").append(freqResult.get(it++)).append("\n");
    }
    fw.write(sb.toString());
    fw.close();
  } catch (IOException ex) {
      System.out.println("Error occured");
  }
}

private void printResult() {
  printFile();
  long duration = endTime - startTime;
  System.out.println("Number of transaction: " + cntTransaction);
  System.out.println("Total number of frequent itemsets: "
                                        + freqItemset.size());
  System.out.println("FP-Growth time: " + duration + "ms");
}
```

2.5. Kết quả và thảo luận

a. Kết quả

Dữ liệu từ file *.xlsx (hình 2-5)* được lọc, chuyển sang định dạng *.csv (hình 2-6)*.

Với mỗi testcase, chương trình chuyển dữ liệu từ định dạng file *.csv* sang định dạng *.inp* (đối với chương trình tự build và SPMF) và *.arff* (đối với weka).

	A	B
1	**InvoiceNo**	**StockCode**
2	C538847	85232B
3	538848	85232B
4	539338	84077
5	539338	22951
6	539338	22417
7	539338	21977
8	539338	84992
9	539338	21212
10	539338	22949
11	539338	21210
12	539338	22536
13	539338	22530
14	539338	22242
15	539338	22489
16	539338	22758
17	539338	22757

Hình 2-5. Excel format

F: > CSD > test4.csv

```
1   C538847,85232B
2   538848,85232B
3   539338,84077
4   539338,22951
5   539338,22417
6   539338,21977
7   539338,84992
8   539338,21212
9   539338,22949
10  539338,21210
11  539338,22536
12  539338,22530
13  539338,22242
14  539338,22489
15  539338,22758
16  539338,22757
```

Hình 2-6. CSV format

```
21108 21977 22325 22371 22377 22467 22556 22626 22932
POST
21260 21715 21716 21791 21889 21906 22099 22489 22492
21452 21731 21788 21789 22232 22378 22492 22779 23084
22045 22139 22708 23231 23236 23293 23295 23296 23494
21210 21212 21216 21977 22242 22326 22328 22348 22417
22077
21731 22328 22467 23084 23480 POST
20712 20713 20718 20719 20724 21731 21933 21981 22356
20704 21260 21906 21977 22348 22907 22909 22950 22952
21917 21918 22139 22491 22551 22554 22720 22961 23191
20725 20726 22326 22328 22352 22779 22993 23206 23254
22467 POST
```

Hình 2-7. Input in .txt format

test4.arff

F: > CSD > test4.arff

```
1   @relation retail
2
3   @attribute 15036 {true}
4   @attribute 15058A {true}
5   @attribute 20704 {true}
6   @attribute 20712 {true}
7   @attribute 20713 {true}
8   @attribute 20718 {true}
9   @attribute 20719 {true}
10  @attribute 20724 {true}
11  @attribute 20725 {true}
12  @attribute 20726 {true}
13  @attribute 20728 {true}
14  @attribute 20973 {true}
15  @attribute 20974 {true}
16  @attribute 20975 {true}
17  @attribute 20981 {true}
```

Hình 2-8. Input in .arff format

Kiểm tra độ chính xác của chương trình triển tự khai giải thuật FP-Growth bằng cách so sánh kết quả của chương trình với kết quả từ 2 công cụ là Weka và SPMF.

Testcase 1 / 7: Tự sinh

Input file: *test1.inp* Kích thước: *nhỏ*

Code output file: *test1.out* Tool output file: *test1_tool.out*

Số giao dịch: $|T| = 5$

Số hạng mục: $|i| = 17$

Độ hỗ trợ nhỏ nhất: $MinSupp = 0.5\%$

So sánh kết quả chương trình tự build với kết quả Weka: 100%.

Input	Code output	Tool output
a c d f g i m p a b c f l m o b f h j o b c k p s a c e f l m n p	a:3 b:3 c:4 f:4 m:3 p:3 a c:3 a f:3 a m:3 c f:3 c m:3 c p:3 f m:3 a c f:3 a c m:3 a f m:3 c f m:3 a c f m:3	a:3 b:3 c:4 f:4 m:3 p:3 a c:3 a f:3 a m:3 c f:3 c m:3 c p:3 f m:3 a c f:3 a c m:3 a f m:3 c f m:3 a c f m:3

```
TEST #1:
CONVERT INPUT FILES
File name:  test1.csv
Converted file: test1.inp
Number of transactions: 5
Number of items: 17
Encode input time: ~21ms
------------------------------
RUN FP-GROWTH
Number of transactions: 5
Number of items: 17
Total number of frequent itemsets: 18
FP-Growth time: ~4ms
------------------------------
COMPARING RESULT
Code output file: test1.out
Tool output file: test1_tool.out
Compare result: 100.00%
Compare time: ~11ms
```

Hình 2-9. Kết quả testcase #1

Testcase 2 / 7: Tự sinh

Input file: *test2.inp* Kích thước: *nhỏ*

Code output file: *test2.out* Tool output file: *test2_tool.out*

Số giao dịch: $|T| = 12$

Số hạng mục: $|i| = 6$

Độ hỗ trợ nhỏ nhất: $MinSupp = 0.3\%$

So sánh kết quả chương trình tự build với kết quả Weka: 100%.

Input	Code output	Tool output
Beans Eggs Yogurt Cheese Milk Yogurt Cheese Eggs Milk Snack Yogurt Cheese Eggs Milk Yogurt Beans Eggs Snack Beans Yogurt Cheese Milk Snack Yogurt Cheese Milk Snack Yogurt Milk Snack Beans Eggs Milk Yogurt Beans Cheese Milk Snack Eggs Snack Yogurt	Eggs:6 Milk:8 Beans:5 Snack:7 Cheese:6 Yogurt:9 Milk Snack:5 Cheese Milk:6 Eggs Yogurt:5 Milk Yogurt:6 Cheese Yogurt:5 Cheese Milk Yogurt:5	Eggs:6 Milk:8 Beans:5 Snack:7 Cheese:6 Yogurt:9 Milk Snack:5 Cheese Milk:6 Eggs Yogurt:5 Milk Yogurt:6 Cheese Yogurt:5 Cheese Milk Yogurt:5

```
TEST #2:
CONVERT INPUT FILES
File name:  test2.csv
Converted file: test2.inp
Number of transactions: 12
Number of items: 6
Encode input time: ~3ms
------------------------------
RUN FP-GROWTH
Number of transactions: 12
Number of items: 6
Total number of frequent itemsets: 12
FP-Growth time: ~2ms
------------------------------
COMPARING RESULT
Code output file: test2.out
Tool output file: test2_tool.out
Compare result: 100.00%
Compare time: ~8ms
```

Hình 2-10. Kết quả testcase #2

Testcase 3 / 7: Tự sinh

Input file: *test3.inp* Kích thước: *trung bình*

Code output file: *test3.out* Tool output file: *test3_tool.out*

Số giao dịch: $|T| = 30$

Số hạng mục: $|i| = 50$

Độ hỗ trợ nhỏ nhất: $MinSupp = 0.2\%$

So sánh kết quả chương trình tự build với kết quả Weka: 100%.

```
TEST #3:
CONVERT INPUT FILES
File name:  test3.csv
Converted file: test3.inp
Number of transactions: 30
Number of items: 50
Encode input time: ~20ms
------------------------------
RUN FP-GROWTH
Number of transactions: 30
Number of items: 50
Total number of frequent itemsets: 70
FP-Growth time: ~11ms
------------------------------
COMPARING RESULT
Code output file: test3.out
Tool output file: test3_tool.out
Compare result: 100.00%
Compare time: ~30ms
```

Hình 2-11. Kết quả testcase #3

Các file dữ liệu input, output của chương trình tự build, output của công cụ trong thư mục *test/testcase/…* của project FP Growth.

Testcase 4 / 7: Dữ liệu hóa đơn bán hàng online của Sweden

Input file: *test4.inp* Kích thước: *trung bình*

Code output file: *test4.out* Tool output file: *test4_tool.out*

Số giao dịch: $|T| = 47$

Số hạng mục: $|i| = 262$

Độ hỗ trợ nhỏ nhất: $MinSupp = 0.05\%$

So sánh kết quả chương trình tự build với kết quả Weka: 99.87%.

```
TEST #4:
CONVERT INPUT FILES
File name:  test4.csv
Converted file: test4.inp
Number of transactions: 47
Number of items: 262
Encode input time: ~9ms
------------------------------
RUN FP-GROWTH
Number of transactions: 47
Number of items: 262
Total number of frequent itemsets: 3169
FP-Growth time: ~46ms
------------------------------
COMPARING RESULT
Code output file: test4.out
Tool output file: test4_tool.out
Compare result: 99.87%
Compare time: ~119ms
```

Hình 2-122-5. Kết quả testcase #4

Các file dữ liệu input, output của chương trình tự build, output của công cụ trong thư mục *test/testcase/…* của project FP Growth.

Testcase 5 / 7: Dữ liệu hóa đơn bán hàng online của Portugal

Input file: *test5.inp* Kích thước: *trung bình*

Code output file: *test5.out* Tool output file: *test5_tool.out*

Số giao dịch: $|T| = 71$

Số hạng mục: $|i| = 706$

Độ hỗ trợ nhỏ nhất: $MinSupp = 0.05\%$

So sánh kết quả chương trình tự build với kết quả Weka: 95.59%.

```
TEST #5:
CONVERT INPUT FILES
File name:  test5.csv
Converted file: test5.inp
Number of transactions: 71
Number of items: 706
Encode input time: ~33ms
------------------------------
RUN FP-GROWTH
Number of transactions: 71
Number of items: 706
Total number of frequent itemsets: 2277
FP-Growth time: ~130ms
------------------------------
COMPARING RESULT
Code output file: test5.out
Tool output file: test5_tool.out
Compare result: 95.59%
Compare time: ~37ms
```

Hình 2-13. Kết quả testcase #5

Các file dữ liệu input, output của chương trình tự build, output của công cụ trong thư mục *test/testcase/…* của project FP Growth.

Testcase 6 / 7: Dữ liệu hóa đơn bán hàng online của France

Input file: *test6.inp* Kích thước: *trung bình*

Code output file: *test6.out* Tool output file: *test6_tool.out*

Số giao dịch: $|T| = 462$

Số hạng mục: $|i| = 1544$

Độ hỗ trợ nhỏ nhất: $MinSupp = 0.05\%$

So sánh kết quả chương trình tự build với kết quả Weka: 88.44%.

```
TEST #6:
CONVERT INPUT FILES
File name:  test6.csv
Converted file: test6.inp
Number of transactions: 462
Number of items: 1544
Encode input time: ~120ms
------------------------------
RUN FP-GROWTH
Number of transactions: 462
Number of items: 1544
Total number of frequent itemsets: 149
FP-Growth time: ~17ms
------------------------------
COMPARING RESULT
Code output file: test6.out
Tool output file: test6_tool.out
Compare result: 88.44%
Compare time: ~5ms
```

Hình 2-64. Kết quả testcase #6

Các file dữ liệu input, output của chương trình tự build, output của công cụ trong thư mục *test/testcase/…* của project FP Growth.

Testcase 7 / 7: Dữ liệu hóa đơn của France

Input file: *test7.inp* Kích thước: *lớn*

Code output file: *test7.out* Tool output file: *test7_tool.out*

Số giao dịch: $|T| = 23495$

Số hạng mục: $|i| = 4066$

Độ hỗ trợ nhỏ nhất: $MinSupp = 0.05\%$

Với kích thước input của test này, khi chạy Apriori để sinh tập phổ biến ở Weka, sẽ bị lỗi *Time limit exceed*. Do đó, nhóm sử dụng công cụ SPMF thay thế.

So sánh kết quả chương trình tự build với kết quả SPMF: 100 %.

```
TEST #7:
CONVERT INPUT FILES
File name:  test7.csv
Converted file: test7.inp
Number of transactions: 23495
Number of items: 4066
Encode input time: ~6555ms
------------------------------
RUN FP-GROWTH
Number of transactions: 23495
Number of items: 4066
Total number of frequent itemsets: 14
FP-Growth time: ~576ms
------------------------------
COMPARING RESULT
Code output file: test7.out
Code output file: test7_tool.out
Compare result: 100.00%
Compare time: ~25ms
```

Hình 2-75. Kết quả testcase #7

Các file dữ liệu input, output của chương trình tự build, output của công cụ trong thư mục *test/testcase/…* của project FP Growth.

Nhận xét về kết quả:

Ở các bộ dữ liệu vừa và nhỏ, chương trình FP-Growth tự triển khai với công cụ Weka đều cho ra kết quả giống nhau tuyệt đối. Khi xử lý các bộ dữ liệu lớn hơn, đã có những sai số xảy ra.

Đặt $r = \frac{|T|}{|I|}$, với T là tập giao dịch, và I là tập các *item* khác nhau trong tất cả các giao dịch. Với $r < 1$, khi r càng lớn, đồng nghĩa với việc số giao dịch càng nhiều, độ đa dạng của các tập phổ biến càng lớn. So với output của công cụ, thuật toán FP-Growth triển khai bởi nhóm sẽ cho tỉ lệ chính xác hơn với r nhỏ, và độ chính xác tỉ lệ nghịch với r. Với $r \geq 1$, số *item* có giới hạn nhất định, số tập phổ biến sinh ra cũng nhỏ gọn hơn, độ chính xác sẽ cao hơn.

Nhận định thấy độ chính xác của kết quả thay đổi phụ thuộc vào các yếu tố:

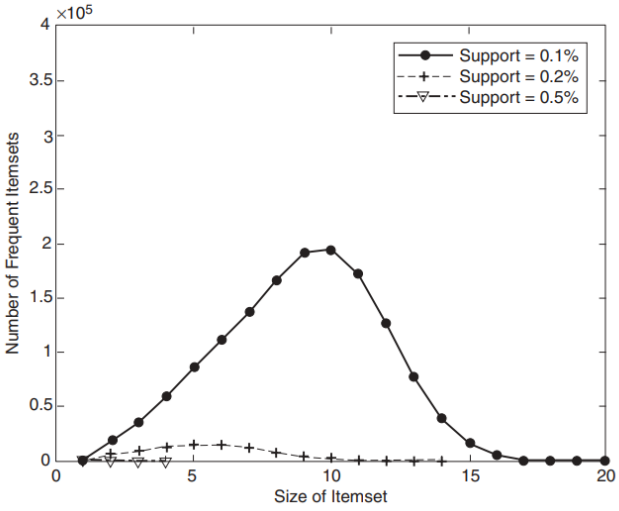
(1) MinSupp: càng nhỏ thì càng có nhiều tập phổ biến được sinh ra, khả năng sai số khi so sánh càng cao.

(2) Số giao dịch: càng tăng thì độ MinSupp tương đối cũng tăng theo.

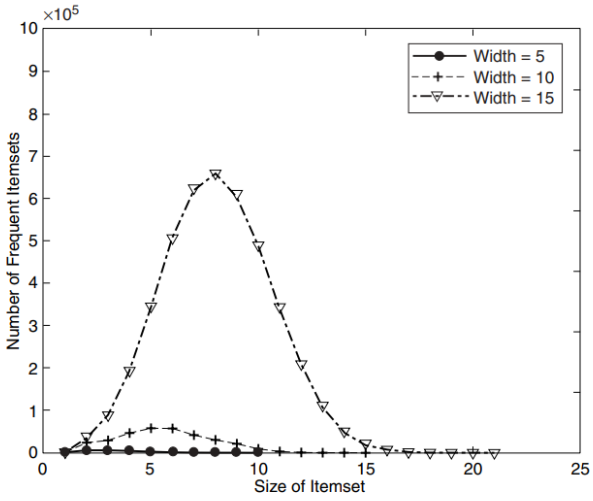
(3) Số item của tất cả các giao dịch

(4) Kích cỡ tối đa của 1 giao dịch

Các yếu tố trên không chỉ ảnh hưởng độ chính xác mà còn ảnh hưởng đến độ phức tạp không gian, thời gian thực thi giải thuật.



Hình 2-16. Ảnh hưởng của MinSupp tới số tập phổ biến



Hình 2-17. Ảnh hưởng của kích cỡ giao dịch tới số tập phổ biến

MinSupp tác động đến độ phức tạp vì MinSupp cảng nhỏ càng nhiều mẫu điều kiện cần tính toán, kích thước và số tập phổ biến cũng tăng lên. Khi số item tăng, không chỉ số tập phổ biến có thể tăng, còn cần nhiều không gian hơn để lưu trữ item cũng như support của chúng, nó còn tăng chi phí nhập xuất dữ liệu. Và khi một giao dịch càng chứa nhiều item thì tiềm năng làm tăng kích thước và số tập phổ biến càng cao.

b. Đánh giá thuật toán

Độ phức tạp về thời gian:

Xây dựng FP-tree:

+ Sắp xếp và đếm các hạng mục trong các giao dịch:

  $O(|T|log|I|)$, T: tập giao dịch,
  I: tập các hạng mục trong giao dịch.

+ Truy ngược FP-Tree để tìm cơ sở mẫu điều kiện:

  $O(|i|h)$, *i:* tập hạng mục phổ biến từ Header Table,
  h: chiều cao tối đa của FP-Tree.

Vậy, độ phức tạp khi xây FP-Tree là $\boldsymbol{O(|i| \times h)}$**.** *(1)*

Khai thác các tập phổ biến trong cơ sở dữ liệu mẫu:

+ Gọi S_{item} là cơ sở dữ liệu mẫu của *item*. Tổng quan, ta sử dụng kĩ thuật trạng thái bit để duyệt từng tập con của S_{item}, nên có độ phức tạp là $O(2^{|S_{item}|})$. *(2)*

+ Trên thực tế, vì S_{item} thường khá nhỏ, nên không ảnh hưởng tới độ phức tạp chung của thuật toán

Từ *(1)* và *(2)*, ta kết luận độ phức tạp của FP-Growth là $\boldsymbol{O(|i| \times h)}$. [5]

Độ phức tạp về không gian:

- O(n), n là số các tác vụ của cơ sở dữ liệu.
- Độ cao của cây được giới hạn bởi kích thước của *itemset* lớn nhất. Trường hợp xấu nhất là khi mỗi giao dịch có một tập hạng mục độc lập, khi đó, không gian cần để lưu trữ cây sẽ lớn hơn không gian lưu cơ sở dữ liệu gốc vì FP-tree cần thêm không gian để lưu liên kết giữa các nút và đếm *support* của mỗi item. [6]

Đặc điểm:

Trích dẫn từ *Data Mining: Conceps and Techniques 2nd Edition (Jiawei Han và Micheline Kamber)* nói rằng:

"Một nghiên cứu về hiệu suất của thuật toán FP-Growth cho thấy nó hiệu quả và linh hoạt có thể co giãn, mở rộng phù hợp để khai thác với cả tập dữ liệu phổ biến dài và ngắn, nhanh hơn thuật toán Apriori về cấp độ, thậm chí nhanh hơn cả thuật toán Tree – Projection."

Khác với thuật toán Apriori cũng là một thuật toán phổ biến để khai phá các tập phổ biến, FP-Growth đề xuất phương án xử lí không cần sinh và kiểm tra tập ứng viên. Cách làm này xây dựng cấu trúc FP-tree để lưu trữ toàn bộ cơ sở dữ liệu, thay đổi việc kiểm tra các mẫu dài thành các khai phá các mẫu ngắn trên cây một cách đệ quy. Làm giảm đáng kể chi phí tìm kiếm. chi phí chủ yếu là đếm và xây dựng cây FP Tree lúc dầu.

Hai ưu điểm của FP-Growth để cải thiện hiệu suất đáng kể so với các thuật toán tìm mẫu phổ biến khác là:

- **Completeness:**
  Sử dụng cấu trúc dữ liệu FP-Tree giúp bảo toàn toàn bộ dữ liệu được sử dụng để khai thác các mẫu phổ biến, không bao giờ thực hiện bất kỳ sửa đổi nào đối với dữ liệu hiện có, cũng như không bao giờ cắt ngắn các mẫu dữ liệu dài của bất kỳ giao dịch nào.
- **Compactness:**
  FP-Growth cho phép loại bỏ các hạng mục bằng cách chỉ lọc ra những mẫu thường xuyên nhất từ FP-Tree. Các mẫu trong FP-Tree được sắp xếp theo thứ tự tần số giảm dần, giảm đáng kể sự phức tạp của quá trình trích xuất các mẫu phổ biến.

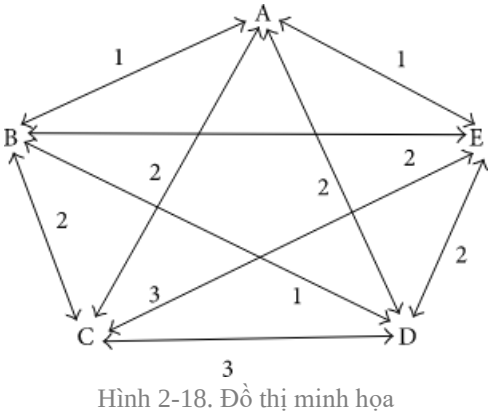
c. Hướng phát triển

Tiếp tục khảo sát và thử nghiệm để tìm ra thiếu sót trong chương trình, cải thiên kết quả so sánh với các công cụ. Tìm hiểu những ứng dụng của việc nghiên cứu thuật toán khai phá luật kết hợp và thực hành thí nghiệm vào các công việc cụ thể. Thử nghiệm các phương pháp khai phá luật phổ biến, luật kết hợp khác. Nghiên cứu các thuật toán, phương pháp khai phá luật kết hợp khác, tiếp tục tìm hiểu, tra cứu tìm ra các phương pháp tối ưu cho qua trình thực thi giải thuật FP-Growth nói riêng và khai phá luật phổ biến nói chung.

Ví dụ cải tiến quá trình build FP-Tree: Như ta thấy, việc sắp xếp thứ tự các *item* giảm dần về tần suất giúp tăng hiệu suất của giải thuật, tuy nhiên, cách làm này chưa thực sự tối ưu nhất. Trường hợp các nút không có chung tiền tố, tức FP-Tree bị rẽ nhánh quá nhiều. Việc rẽ nhánh nhiều như vậy sẽ giảm độ hiệu quả của thuật toán, cũng như không tận dụng được lợi thế các mối quan hệ giữa các *item* trong cơ sở dữ liệu. Ở đây, nhóm đề xuất sử dụng một số cải tiến / giải thuật khác như:

Painting-grow: Là một cải tiến của FP-Growth, nhưng không sử dụng cây tiền tố làm cấu trúc dữ liệu chính, mà sử dụng một bảng kích cỡ $|I| \times |I|$, gọi là M, với I là tập các hạng mục. Với mỗi giao dịch, gọi $S = \{i_1, i_2, i_3, \dots, i_k\}$, ta sẽ thêm vào đồ thị có hướng các cạnh $\{i_1, i_2\}, \{i_1, i_3\}, \dots, \{i_1, i_k\}, \dots, \{i_{k-1}, i_k\}$, là các tập con kích cỡ 2 của S. Nếu cạnh đã tồn tại, tăng số đếm bằng cách cộng thêm 1 vào bảng ma trận $M[i_u][i_v]$. Hình minh họa ở dưới.

Bằng cách này, ta dễ dàng quản lý mối quan hệ giữa các *item*, tạo ra một cấu trúc dữ liệu chặt chẽ hơn. Với mỗi *item*, có thể dễ dàng tạo cơ sở dữ liệu mẫu và khai phá các tập phổ biến, kết hợp trên đó.



Hình 2-18. Đồ thị minh họa

Chương 3. KẾT LUẬN

Qua quá trình thực hiện nghiên cứu, tìm hiểu và phân tích cũng như triển khai chủ đề chính của bài tập là giải thuật FP-Growth, hiện tại nhóm đã đạt được những mục tiêu sau:

+ Tìm hiểu bài toán phân tích tìm tập luật phổ biến, triển khai thành công giải thuật FP-Growth.
+ Khảo sát và thực hiện các giải thuật khai phá luật phổ biến trên tập dữ liệu hóa đơn bán lẻ online của UK, France, Portual, Sweden.
+ Qua đó xác thực lại tính đúng đắn của giải thuật và phần trăm chính xác của chương trình tự triển khai.

Qua những phần báo cáo trình bày ở các mục phía trên, nhóm đã thảo luận chi tiết về thuật toán FP-Growth, đóng vai trò như giải pháp thay thế hiệu quả cho các thuật toán nổi tiếng khác như Apriori, ECLAT. Có thể nhận thấy giải thuật FP-Growth có những phần khác biệt, cung cấp khả năng tối ưu hóa hiệu suất ở mức độ thuật toán nhiều hơn, từ đó giảm thời gian thực thi cũng như chi phí bộ nhớ, và tạo tiền đề dể khai thác các quy tắc, các tri thức khác từ cơ sở dữ liệu được nén một cách nhanh chóng hơn, nâng cao hiệu suất. Việc nghiên cứu, phân tích thuật toán này không chỉ mang ý nghĩa học thuật mà còn có những ứng dụng thực tiễn với các tác vụ của các lĩnh vực khác nhau như: marketing, phân tích quyết định, quản trị kinh doanh, các ngành khoa học khác như y hóa sinh,…

Với đề tài này, mỗi cá nhân của nhóm đã thực sự được trải nghiệm, thực hành những kiến thức mới, hấp dẫn, có giá trị cao, có cơ hội trau dồi thêm các kĩ năng liên quan hỗ trợ cho chuyên môn bản thân. Bài học rút ra qua quá trình thực hiện đề tài:

Đăng Lộc: biết được quy trình chuyên môn khi thực hiện Data mining, các định nghĩa, khái niệm cơ bản khi bước đầu tiếp cận với Data mining, các chủ đề của khai thác dữ liệu và ứng dụng của nó vào thực tiễn; phát triển kĩ năng, kinh nghiệm tìm kiếm các nguồn tài liệu học thuật để đọc, tham khảo và chọn lọc ứng dụng; có thể xử lý dữ liệu thu gom được từ các nguồn quản trị cơ sở dữ liệu, các kho dữ liệu được chia sẻ, tiền kì để dữ liệu đó có thể khai thác qua các giải thuật, mã nguồn; thêm kinh nghiệm làm việc với ngôn ngữ Java trong nhiều tác vụ khác nhau.

Vĩ Khang: Có cái nhìn sâu hơn trong thế giới Data mining, hiểu được và hiện thực hóa được thuật toán FP-Growth, và biết thêm các kiến thức toán học cũng như cấu trúc dữ liệu mới Học cách trình bày một bản báo cáo về thuật toán, cũng như nâng cao khả năng làm việc nhóm Biết được cách thức sử dụng các tool data mining khác nhau như WEKA, SPMF,... trên các dataset khác nhau và tham khảo các thư viện để so sánh kết quả.

Trên đây là toàn bộ báo cáo của nhóm 8 – Nguyễn Vĩ Khang, Nguyễn Đăng Lộc đối với đề bài triển khai giải thuật FP-Growth. Do thời gian cũng như trình độ của nhóm còn hạn chế nên bản trình bày cũng như kiến thức có thể chưa được tối ưu, hoàn toàn chính xác và có những thiếu sót, khuyết điểm, kính mong được thầy

thông cảm và góp ý nhận xét cho bản báo cáo của nhóm được hoàn chỉnh hơn và nhóm có thể rút ra được những kinh nghiệm quý giá cho bản thân.

TỰ ĐÁNH GIÁ

Nội dung	Điểm	Ghi chú
Giới thiệu về bài toán (0.25 đ)	**0.25**	
Mô tả cấu trúc dữ liệu (1.25 đ)	**1.25**	
Sơ đồ giải thuật (1.5 đ)	**1.5**	
Hiện thực (5 đ)	**4.75**	
Kết quả và thảo luận (0.5 đ)	**0.25**	
Điểm nhóm (0.75 đ)	**0.75**	
Các điều rút ra cho bản thân (0.25 đ)	**0.25**	
Báo cáo đúng theo mẫu (0.5 đ)	**0.5**	
Tổng điểm	**9.5**	

TÀI LIỆU THAM KHẢO

[1] J. Han, Data Mining: Concepts and Techniques, vol. 5, San Francisco, CA 94111: Diane Cerra, 2000.

[2] D. Loshin, Business Intelligence (Second Edition) - The Savvy Manager's Guide, Maryland, United States, 2013.

[3] M. S. A. K. V. K. Pang-Ning Tan, "Introduction to Data Mining, 2nd Edition," in *Association Analysis: Basic Concepts and Algorithms*, Michigan State University, Pearson, 2019, pp. 329-335.

[4] N. R. T. Trung, "Trie," 2016. [Online]. Available: https://vnoi.info/wiki/algo/data-structures/trie.md.

[5] A. V. Ratz, "CodeProject," 20 July 2019. [Online]. Available: https://www.codeproject.com/Articles/5162780/Association-Rules-Learning-ARL-Part-2-FP-Growth-Alg.

[6] A. L. Thabet Slimani, "Efficient Analysis of Pattern and Association Rule Mining Approaches," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 6, pp. 70-81, 2014.