BÁO CÁO CÔNG VIỆC

**Công việc số:** 54

**Tên công việc:** Lập trình thuật toán FP-Growth và Ghép 2 thuật toán vào Project tìm luật kết hợp hoàn chỉnh.

**Người thực hiện:** Hạ Quang Dũng

**Ngày bắt đầu:** 25/09/2024

**Ngày kết thúc:** 12/10/2024

# Mục lục

[**Mục lục 1**](#_mk7b1h3jycpl)

[**Tổng quan 2**](#_xwz3kxggpmyt)

[**1. Algorithms (Thuật toán) 2**](#_cm8ksuqmglf1)

[1.1. Apriori 3](#_rviup1fu7k6u)

[1.2. FPGrowth 5](#_akvbjs8zvues)

[1.2.1. Item 5](#_rbm8nec3d1vh)

[1.2.2. Node 5](#_49ac5mwisdzz)

[1.2.3. Tree 6](#_qohlf6q28c0y)

[1.2.4. FPGrowth 7](#_nojkuwvdrugf)

[**2. Common (Chung) 11**](#_pn8r1mijmmy)

[2.1. AssociationRule 11](#_mrpzv85kwdpg)

[2.2. Helper 12](#_s64lqpla7d8g)

[2.3. ItemSet 15](#_9a24nai4h39d)

[**3. Main (Chính) 16**](#_ns5bkok0g2nd)

[**4. Hướng dẫn sử dụng và kết quả thực hiện 18**](#_2pguh6omz5v3)

**Tài liệu dự án AssociationRuleFinder**

# Tổng quan

Dự án này gồm 3 phần chính:

* Algorithms
* Common
* Main

Mỗi phần đóng một vai trò quan trọng trong việc triển khai khai phá luật kết hợp. Tài liệu này sẽ giải thích chi tiết vai trò của từng phần, các thành phần chính và cách chúng liên quan với nhau.

# 1. Algorithms (Thuật toán)

Phần **Algorithms** chứa các thuật toán được sử dụng để khai phá luật kết hợp. Các thành phần chính trong dự án này bao gồm:

* **Thư mục Apriori**:
  + **Apriori.cs**: Chứa triển khai của thuật toán Apriori, bao gồm logic để tìm các tập phổ biến và tạo ra các luật kết hợp từ tập dữ liệu.
* **Thư mục FPGrowth:**
  + **Item.cs**: Định nghĩa lớp Item, có thể đại diện cho một phần tử trong dữ liệu giao dịch, lưu trữ thông tin về tên phần tử và số lượng.
  + **Node.cs**: Định nghĩa lớp Node, có thể đại diện cho một nút trong cây FP (Frequent Pattern Tree), chứa thông tin về phần tử và các liên kết tới các nút con.
  + **Tree.cs**: Định nghĩa lớp Tree, chịu trách nhiệm xây dựng và quản lý cây FP, bao gồm việc thêm các phần tử và xây dựng cấu trúc cây từ dữ liệu giao dịch.
  + **FPGrowth.cs**: Chứa triển khai của thuật toán FP-Growth, thực hiện khai phá các tập phổ biến bằng cách sử dụng cây FP, từ đó tạo ra các luật kết hợp.

## **1.1. Apriori**

| using Common; using System.Collections.Generic; using System.Linq;  namespace Algorithms.Apriori {  public class Apriori  {  private List<string> transactions; // Danh sách giao dịch đã được tiền xử lý  private List<string> distinctValues; // Các giá trị mục riêng biệt  private int minSupportCount; // Ngưỡng hỗ trợ tối thiểu   // 1. Constructor: Khởi tạo với danh sách giao dịch và ngưỡng hỗ trợ tối thiểu  public Apriori(List<string> transactions, int minSupport)  {  // Tiền xử lý danh sách giao dịch để chuẩn hóa định dạng  this.transactions = transactions;  minSupportCount = minSupport;  SetDistinctValues();  }   // 2. Thiết lập danh sách các giá trị duy nhất từ các giao dịch đã tiền xử lý  private void SetDistinctValues()  {  distinctValues = transactions  .SelectMany(t => t.Split(',')) // Tách từng mục trong giao dịch (dựa trên dấu ',')  .Where(v => !string.IsNullOrWhiteSpace(v))  .Distinct()  .OrderBy(a => a)  .ToList();  }   // 3. Tìm tập mục phổ biến theo độ dài và ngưỡng hỗ trợ  public ItemSet GetItemSet(int length, int support)  {  var itemSet = new ItemSet { Support = support, Label = "L" + length };  var combinations = GetPermutations(distinctValues, length).Select(p => p.ToList());   foreach (var combination in combinations)  {  // Kiểm tra xem các mục của tập hợp có xuất hiện trong tất cả các giao dịch không  int count = transactions.Count(t => combination.All(item => t.Split(',').Contains(item)));  if (count >= support)  itemSet[combination] = count;  }   return itemSet;  }   // 4. Helper: Tìm tất cả các tổ hợp có độ dài count từ danh sách đầu vào  public static IEnumerable<IEnumerable<T>> GetPermutations<T>(IEnumerable<T> items, int count)  {  return count == 1  ? items.Select(item => new T[] { item })  : items.SelectMany((item, i) => GetPermutations(items.Skip(i + 1), count - 1).Select(p => new[] { item }.Concat(p)));  }   // 5. Tìm tất cả các tập mục phổ biến  public List<List<string>> GetFrequentItemSet()  {  int length = 1;  var frequentItemSets = new List<List<string>>();   while (true)  {  var itemSet = GetItemSet(length, minSupportCount);  if (!itemSet.Any()) break; // Nếu không còn tập mục phổ biến nào, dừng vòng lặp  frequentItemSets.AddRange(itemSet.Keys);  length++;  }   return frequentItemSets;  }  } } |
| --- |

## 1.2. FPGrowth

### **1.2.1. Item**

| namespace Algorithms.FPGrowth {  public class Item  {  public string Name { get; set; } // Tên phần tử  public int Count { get; set; } // Số lần phần tử xuất hiện  public Item(string name)  {  Name = name;  Count = 0;  }  public Item(string name, int count)  {  this.Name = name;  this.Count = count;  }  } } |
| --- |

### 1.2.2. Node

| using System.Collections.Generic;  namespace Algorithms.FPGrowth {  public class Node  {  public Dictionary<string, Node> Children { get; set; } // Danh sách các nút con  public Item ItemData { get; set; } // Dữ liệu của phần tử tại nút này  public int Count { get; set; } // Số lần xuất hiện của phần tử  public Node Parent { get; set; } // Thêm thuộc tính để chỉ định nút cha  public List<Node> NodeLink { get; set; } // Danh sách liên kết nút trỏ về các phần tử tương ứng  public Node()  {  Children = new Dictionary<string, Node>();  ItemData = null;  Count = 0;  Parent = null;  NodeLink = new List<Node>();  }  } } |
| --- |

### 1.2.3. Tree

| using System; using System.Collections.Generic;  namespace Algorithms.FPGrowth {  public class Tree  {  public Node Root { get; private set; } // Gốc của cây  private readonly int \_minSupportCount; // Ngưỡng hỗ trợ tối thiểu   public Tree(int minSupport)  {  Root = new Node(); // Khởi tạo nút gốc  \_minSupportCount = minSupport; // Thiết lập ngưỡng hỗ trợ tối thiểu  }  public void BuildTree(ref List<string> elements)  {  Node currentNode = Root;   foreach (var element in elements)  {  if (currentNode.Children.ContainsKey(element))  {  // Nếu phần tử đã tồn tại, tăng số lần xuất hiện  currentNode.Children[element].Count++;  currentNode = currentNode.Children[element];  }  else  {  // Nếu phần tử chưa tồn tại, tạo nút mới  var newNode = new Node { ItemData = new Item(element), Count = 1 };  currentNode.Children[element] = newNode;  currentNode = newNode;  }  }  }  public void PrintTree(Node node, string indent = "")  {  if (node == null) return;   foreach (var child in node.Children)  {  Console.WriteLine($"{indent}{child.Key} ({child.Value.Count})");  PrintTree(child.Value, indent + " "); // Đệ quy để in các nút con  }  }  public void FindPaths(Node node, List<Item> currentPattern, List<List<Item>> allPaths)  {  if (node == null) return;   foreach (var child in node.Children)  {  var newPattern = new List<Item>(currentPattern)  {  new Item(child.Key, child.Value.Count) // Thêm phần tử vào đường đi mới  };   allPaths.Add(newPattern); // Lưu đường đi vào danh sách  FindPaths(child.Value, newPattern, allPaths); // Tiếp tục đệ quy với các con  }  }  } } |
| --- |

### 1.2.4. FPGrowth

| using Common; using System.Linq; using System.Collections.Generic;  namespace Algorithms.FPGrowth {  public class FPGrowth  {  private readonly int minSupportCount;  private List<string> transactions;  public FPGrowth(List<string> transactions, int support)  {  minSupportCount = support;  this.transactions = transactions;   }  public List<List<string>> GetFrequentItemsets()  {  // Tạo C1 và lọc L1  var c1ItemList = GenerateC1(transactions);  var l1ItemList = PruneL1(c1ItemList);  SortItemsDescending(l1ItemList);   // Xây dựng cây FP-tree từ dữ liệu đã lọc  var fpTree = BuildFPTree(transactions, l1ItemList);   // Lấy danh sách các đường đi từ cây FP  var pathListByItem = GetPathsFromTree(fpTree);   // Tạo cơ sở mẫu điều kiện từ các đường đi  var conditionalPatternBase = GetConditionalPatternBase(pathListByItem, l1ItemList);   // Khai thác tập mục phổ biến  var frequentItemsets = new List<List<string>>();  var uniqueCombinations = new HashSet<string>();   foreach (var itemList in conditionalPatternBase)  {  for (int k = 0; k < itemList.Count; k++)  {  var combinations = Helper.GetCombinations(itemList, k + 1);  foreach (var combination in combinations)  {  var combinationKey = string.Join(", ", combination.Select(i => i.Name));   if (uniqueCombinations.Add(combinationKey))  {  frequentItemsets.Add(combination.Select(i => i.Name).ToList());  }  }  }  }   // Lọc các tập phổ biến không đủ số lần xuất hiện  frequentItemsets = frequentItemsets  .Where(itemset => Helper.CountItemset(itemset, transactions) >= minSupportCount)  .Distinct()  .OrderBy(itemSet => itemSet.Count) // Sắp xếp theo độ dài  .ThenBy(itemSet => string.Join("", itemSet.OrderBy(i => i))) // Sắp xếp theo thứ tự chữ cái  .ToList();   return frequentItemsets;  }  private List<Item> GenerateC1(List<string> database)  {  var itemSets = new Dictionary<string, Item>();   foreach (var transaction in database)  {  foreach (var item in transaction.Split(',').Select(item => item.Trim()))  {  if (!itemSets.TryGetValue(item, out var existingItem))  {  existingItem = new Item(item);  itemSets[item] = existingItem;  }  existingItem.Count++;  }  }   return itemSets.Values.ToList();  }  private List<Item> PruneL1(List<Item> c1)  {  return c1.Where(item => item.Count >= minSupportCount).ToList();  }  private void SortItemsDescending(List<Item> itemList)  {  itemList.Sort((x, y) => y.Count == x.Count ? x.Name.CompareTo(y.Name) : y.Count.CompareTo(x.Count));  }   private Tree BuildFPTree(List<string> database, List<Item> l1ItemList)  {  var tree = new Tree(minSupportCount);   foreach (var transaction in database)  {  var transactionItems = l1ItemList  .Where(item => transaction.Contains(item.Name))  .Select(item => item.Name)  .ToList();   if (transactionItems.Count > 0)  {  tree.BuildTree(ref transactionItems);  }  }   return tree;  }  private List<List<Item>> GetPathsFromTree(Tree tree)  {  var pathListByItem = new List<List<Item>>();  tree.FindPaths(tree.Root, new List<Item>(), pathListByItem);  return pathListByItem;  }  private List<List<Item>> GetConditionalPatternBase(List<List<Item>> pathList, List<Item> l1ItemList)  {  var conditionalPatternBase = new List<List<Item>>();  foreach (var l1Item in l1ItemList)  {  var itemCounts = new Dictionary<string, Item>();  foreach (var path in pathList)  {  if (l1Item.Name == path.Last().Name)  {  foreach (var pathItem in path)  {  if (!itemCounts.TryGetValue(pathItem.Name, out var existingItem))  {  existingItem = new Item(pathItem.Name, path.Last().Count);  itemCounts[pathItem.Name] = existingItem;  }  else  {  existingItem.Count += path.Last().Count;  }  }  }  }  var filteredPath = itemCounts.Values.Where(i => i.Count >= minSupportCount).ToList();  if (filteredPath.Count > 0)  {  conditionalPatternBase.Add(filteredPath);  }  }   return conditionalPatternBase;  }  } } |
| --- |

# 2. Common (Chung)

Phần **Common** bao gồm các tiện ích và lớp dùng chung giữa các phần khác của dự án. Các tệp chính trong phần này bao gồm:

* **AssociationRule.cs**: Định nghĩa các lớp hoặc phương thức để biểu diễn và xử lý các luật kết hợp.
* **Helper.cs**: Chứa các hàm hoặc phương thức tiện ích, giúp hỗ trợ xử lý dữ liệu hoặc các hoạt động khác cho toàn bộ dự án.
* **ItemSet.cs**: Định nghĩa các cấu trúc hoặc lớp biểu diễn **ItemSet** (tập hợp các phần tử), một thành phần quan trọng trong khai phá tập phổ biến.

## 2.1. AssociationRule

| namespace Common {  public class AssociationRule  {  public string Label { get; set; } // Luật theo dạng "A => B"  public double Support { get; set; } // Hỗ trợ của luật  public double Confidence { get; set; } // Độ tin cậy của luật  } } |
| --- |

## 2.2. Helper

| using System; using System.Collections.Generic; using System.Linq;  namespace Common {  public static class Helper  {  // Tiền xử lý dữ liệu giao dịch: Chuẩn hóa và chuyển về chuỗi với dấu phân cách ","  public static List<string> PreprocessDatabase(List<string> database)  {  return database.Select(transaction => string.Join(",", transaction.Split(' ').Select(item => item.Trim()))).ToList();  }   #region Tìm luật kết hợp  // Lấy tất cả các tổ hợp (combinations) của một danh sách với số lượng phần tử k  public static List<List<T>> GetCombinations<T>(List<T> list, int k)  {  var result = new List<List<T>>();  GetCombinations(list, new List<T>(), k, 0, result);  return result;  }   // Đệ quy để tạo tổ hợp  private static void GetCombinations<T>(List<T> list, List<T> tempList, int k, int start, List<List<T>> result)  {  if (tempList.Count == k)  {  result.Add(new List<T>(tempList));  return;  }   for (int i = start; i < list.Count; i++)  {  tempList.Add(list[i]);  GetCombinations(list, tempList, k, i + 1, result);  tempList.RemoveAt(tempList.Count - 1);  }  }   // Đếm số lần xuất hiện của một tập hợp item trong danh sách giao dịch  public static int CountItemset(List<string> itemset, List<string> transactions)  {  return transactions.Count(transaction => itemset.All(item => transaction.Split(',').Select(i => i.Trim()).Contains(item)));  }   // Tìm luật kết hợp từ các tập phổ biến  public static List<AssociationRule> GetAssociationRules(List<List<string>> frequentItemsets, List<string> transactions, int minSup)  {  var rules = new List<AssociationRule>();   foreach (var itemset in frequentItemsets)  {  for (int k = 1; k < itemset.Count; k++)  {  var combinations = GetCombinations(itemset, k);   foreach (var combination in combinations)  {  var consequent = itemset.Except(combination).ToList();  var rule = new AssociationRule  {  Label = $"{string.Join(", ", combination)} => {string.Join(", ", consequent)}"  };   // Tính toán Support  double support = (double)CountItemset(itemset, transactions) / transactions.Count;  if (support >= (double)minSup / transactions.Count)  {  rule.Support = Math.Round(support \* 100, 2);   // Tính toán Confidence  int combinationCount = CountItemset(combination, transactions);  rule.Confidence = combinationCount == 0 ? 0 : Math.Round((double)CountItemset(itemset, transactions) \* 100 / combinationCount, 2);   rules.Add(rule);  }  }  }  }   // Sắp xếp luật kết hợp theo Support và Confidence giảm dần  return rules.OrderByDescending(r => r.Support).ThenByDescending(r => r.Confidence).ToList();  }  #endregion   #region Hiển thị  // Hiển thị các tập phổ biến  public static void PrintFrequentItemsets(List<List<string>> frequentItemsets, List<string> transactions, int minSup)  {  var groupedByLength = frequentItemsets  .GroupBy(itemset => itemset.Count)  .OrderBy(g => g.Key)  .ToList();   foreach (var group in groupedByLength)  {  Console.WriteLine($"\nFrequent ItemSet (Length = {group.Key}, Support >= {minSup}):");   var sortedItemsets = group  .Select(itemset => itemset.OrderBy(item => item).ToList())  .OrderBy(itemset => string.Join(",", itemset))  .ToList();   foreach (var itemset in sortedItemsets)  {  int count = CountItemset(itemset, transactions);  if (count >= minSup)  {  Console.WriteLine($"Item: {string.Join(", ", itemset)}, Count: {count}");  }  }  }  }   // Hiển thị các luật kết hợp  public static void PrintAssociationRule(List<AssociationRule> rules)  {  Console.WriteLine("\nQuy tắc kết hợp:");  foreach (var rule in rules)  {  Console.WriteLine($"Rule: {rule.Label}, Support: {rule.Support}%, Confidence: {rule.Confidence}%");  }  }  #endregion  } } |
| --- |

## 2.3. ItemSet

| using System.Collections.Generic;  namespace Common {  public class ItemSet : Dictionary<List<string>, int>  {  public string Label { get; set; }  public int Support { get; set; }  } } |
| --- |

### 

# 3. Main (Chính)

Phần **Main** là điểm bắt đầu của dự án. Nó chứa logic chính của ứng dụng và các cấu hình quan trọng:

* **App.config**: Tệp cấu hình của ứng dụng, có thể chứa các cài đặt về môi trường chạy, đường dẫn dữ liệu hoặc các tham số của thuật toán.
* **Program.cs**: Tệp thực thi chính, chứa phương thức Main, nơi bắt đầu thực hiện chương trình. Tệp này có nhiệm vụ xử lý dữ liệu đầu vào, gọi các thuật toán và xuất kết quả.

| using Algorithms.Apriori; using Algorithms.FPGrowth; using Common; using System; using System.Collections.Generic; using System.Diagnostics; using System.Text;  namespace Main {  internal class Program  {  static void Main(string[] args)  {  Console.OutputEncoding = Encoding.UTF8;   // Example usage of the new function  List<string> transactions1 = new List<string>  {  "f a c d g i m p",  "a b c f l m o",  "b f h j o",  "b c k s p",  "a f c e l p m n",  };   int supportThreshold = 3; // Ngưỡng hỗ trợ tối thiểu  var transactionsPreprocessed = Helper.PreprocessDatabase(transactions2);  List<List<string>> frequentItemSets = null;   // Vòng lặp cho phép người dùng chọn thuật toán hoặc thoát  Console.Clear(); // Xóa màn hình để giao diện sạch hơn mỗi lần chọn  Console.WriteLine("Chọn thuật toán để chạy:");  Console.WriteLine("1. FP-Growth");  Console.WriteLine("2. Apriori");  Console.WriteLine("3. Thoát chương trình");  Console.Write("Nhập lựa chọn của bạn (1, 2, hoặc 3): ");  string choice = Console.ReadLine();   Stopwatch stopwatch = new Stopwatch();  stopwatch.Start();  switch (choice)  {  case "1":  // Chạy thuật toán FP-Growth  Console.WriteLine("\nChạy thuật toán FP-Growth...");  FPGrowth fpGrowth = new FPGrowth(transactionsPreprocessed, supportThreshold);  frequentItemSets = fpGrowth.GetFrequentItemsets();   break;   case "2":  // Chạy thuật toán Apriori  Console.WriteLine("\nChạy thuật toán Apriori...");  Apriori apriori = new Apriori(transactionsPreprocessed, supportThreshold);  frequentItemSets = apriori.GetFrequentItemSet();  break;   case "3":  // Thoát chương trình  Console.WriteLine("Kết thúc chương trình.");  return; // Thoát khỏi chương trình   default:  Console.WriteLine("Lựa chọn không hợp lệ. Vui lòng thử lại.");  break;  }   Helper.PrintFrequentItemsets(frequentItemSets, transactionsPreprocessed, supportThreshold);   var rules = Helper.GetAssociationRules(frequentItemSets, transactionsPreprocessed, supportThreshold);  Helper.PrintAssociationRule(rules);  stopwatch.Stop(); // Dừng tính thời gian  Console.WriteLine($"\n\nThời gian thực thi: {stopwatch.Elapsed} ms");   // Sau khi chạy thuật toán, yêu cầu người dùng nhấn phím để tiếp tục hoặc thoát  Console.WriteLine("\nNhấn phím bất kỳ để tiếp tục...");  string input = Console.ReadKey().KeyChar.ToString().ToLower();  }  } } |
| --- |

# 4. Hướng dẫn sử dụng và kết quả thực hiện

Dự án này được thiết kế để khai phá các luật kết hợp từ dữ liệu giao dịch bằng hai thuật toán chính: **Apriori** và **FP-Growth**. Phần Common cung cấp các lớp tiện ích để xử lý dữ liệu, trong khi phần Main đóng vai trò là phần thực thi chính của ứng dụng.

1. Dữ liệu đầu vào: Dữ liệu giao dịch được đưa vào ứng dụng thông qua các tệp hoặc thông qua cấu hình trong **App.config.**
2. Thực thi: Phương thức chính trong Program.cs sẽ khởi động thuật toán được chọn (**Apriori hoặc FP-Growth**) để xử lý dữ liệu đầu vào.
3. Kết quả: Kết quả là các luật kết hợp hoặc tập phổ biến sẽ được tạo ra và có thể xuất ra file hoặc hiển thị trên console.

**Dữ liệu mẫu**

| "f a c d g i m p",  "a b c f l m o",  "b f h j o",  "b c k s p",  "a f c e l p m n", |
| --- |

**Kết quả thực hiện (Tính theo thời gian thực thi)**

| **Số lượng bộ dữ liệu** | **Thuật toán FP Growth** | **Thuật toán Apriori** |
| --- | --- | --- |
| Dữ liệu mẫu (minsup = 3) | 00:00:00.0644157 ms | 00:00:00.0864415 ms |
| 500 giao dịch (minsup = 3) | 00:00:00.0901262 ms | 00:00:09.4156500 ms |
| 1000 giao dịch (minsup = 4) | 00:00:00.4614085 ms | 00:00:20.8943747 ms |
| 1500 giao dịch (minsup = 5) | 00:00:00.6167843 ms | 00:00:33.5613672 ms |
| 2000 giao dịch (minsup = 6) | 00:00:00.9252857 ms | 00:00:45.4635371 ms |
| 3000 giao dịch (minsup = 8) | 00:00:01.1607066 ms | 00:01:01.5352251 ms |
| 5000 giao dịch (minsup = 12) | 00:00:01.6537924 ms | 00:01:50.9322245 ms |
| 7000 giao dịch (minsup = 16) | 00:00:02.2031810 ms | 00:02:46.0204905 ms |
| 10000 giao dịch (minsup = 22) | 00:00:02.1629009 ms | 00:03:46.8356674 ms |
| 15000 giao dịch (minsup = 32) | 00:00:03.8292306 ms | 00:06:15.0153820 ms |
| 20000 giao dịch (minsup = 42) | 00:00:04.5443467 ms | 00:07:28.6409837 ms |
| 30000 giao dịch (minsup = 62) | 00:00:09.0921478 ms | 00:11:54.1364758 ms |
| 50000 giao dịch (minsup = 102) | 00:00:11.8927231 ms | Quá lâu để xử lý |
| 100000 giao dịch (minsup = 202) | 00:00:47.7117791 ms | Quá lâu để xử lý |
| 500000 giao dịch (minsup = 1002) | 00:03:58.5601073 ms | Quá lâu để xử lý |
| 1000000 giao dịch (minsup = 2002) | Quá độ dài cho phép | Quá lâu để xử lý |

**Nhận xét về Kết quả Thực hiện**

1. Hiệu suất tổng quát:
   * FP-Growth hoạt động nhanh hơn đáng kể so với Apriori, cho thấy FP-Growth hiệu quả hơn trong xử lý dữ liệu lớn.
2. Thời gian thực thi:
   * Với 500 giao dịch, FP-Growth mất 00:00:00.1580784 ms trong khi Apriori mất 00:01:26.0174437 ms. Sự chênh lệch này gia tăng khi số lượng giao dịch tăng.
   * Ở mức 2000 giao dịch, FP-Growth chỉ mất 00:00:12.7477830 ms, trong khi Apriori mất tới 00:06:00.0703014 ms.
3. Tác động của số lượng giao dịch:
   * Khi số lượng giao dịch tăng, thời gian thực thi của Apriori tăng nhanh hơn so với FP-Growth, chứng tỏ sự kém hiệu quả của Apriori với dữ liệu lớn.

**Kết luận**

Thuật toán FP-Growth là lựa chọn tối ưu cho khai phá luật kết hợp trong tập dữ liệu lớn nhờ hiệu suất và tốc độ vượt trội so với Apriori. Sử dụng FP-Growth giúp tiết kiệm thời gian và tài nguyên hơn.