BÁO CÁO CÔNG VIỆC

**Công việc số:** 55

**Mô tả công việc:** Tối ưu lại thuật toán FP Growth để không đếm lại mức phổ biến trên giao dịch

**Người thực hiện:** Hạ Quang Dũng

**Ngày bắt đầu:** 13/10/2024

**Ngày kết thúc:** 31/10/2024

Mục lục

[**1. Thuật toán tính độ phổ biến của một tập mục 2**](#_u7rqik8ngayj)

[**1. Trình bày thuật toán 2**](#_1rphde2khumv)

[1.1. Mục tiêu 2](#_jnc1y27f4vos)

[1.2. Ý tưởng chính 2](#_lxo6en5sm3wd)

[1.3. Các bước của thuật toán 2](#_98zd35hef2l3)

[**2. Triển khai thuật toán 3**](#_qk6j6prgz4vv)

[**3. Cập nhật một số hàm liên quan 4**](#_r2wgq68ghy4t)

[3.1. Hàm tìm tập phổ biến 4](#_gqjv9mtfuyu)

[3.2. Hàm in tập phổ biến 6](#_l67eof95rh35)

[3.3. Hàm sinh luật dựa trên tập phổ biến mới 7](#_rbiqkb2oj14t)

[**4. Kết quả thực hiện 8**](#_u9z30k58dvc4)

# 

# 1. Thuật toán tính độ phổ biến của một tập mục

Theo phương pháp cũ, ta đang lưu tập phổ biến dạng List<List<string>>, do đó để tính được độ phổ biến của tập mục ta cần tính đếm lại trên toàn bộ giao dịch với mỗi tập mục. Việc đếm lại đó dẫn đến thời gian xử lý của thuật toán bị chậm. Để có thể tối ưu hơn về thời gian xử lý, ta sẽ lưu danh sách tập phổ biến dưới dạng List<ItemSet>.

Trong đó, lớp ItemSet chứa các thuộc tính sau:

* Label: Tên tập phổ biến
* Support: Số lần xuất hiện của tập phổ biến

Ta sẽ đếm tập phổ biến trên cây để lưu vào support của ItemSet (*Xem chi tiết tại* [*mục 1.3*](#_98zd35hef2l3)). Như vậy ta sẽ ko cần đếm lại nhiều lần, và số lượng node cần đếm cũng sẽ giảm hơn so với giao dịch.

# 1. Trình bày thuật toán

## 1.1. Mục tiêu

Thuật toán này nhằm đếm số lần xuất hiện của một tập mục (pattern) trong một cây dữ liệu có cấu trúc cây, trong đó mỗi nút của cây chứa dữ liệu và có thể có các nút con. Mục tiêu của thuật toán là duyệt qua tất cả các nút trong cây để xác định xem có bao nhiêu đường dẫn phù hợp với tập mục cần tìm.

## 1.2. Ý tưởng chính

Thuật toán sử dụng phương pháp đệ quy để duyệt qua cây dữ liệu và so sánh các phần tử trong tập mục (pattern) với các nút của cây. Nếu một nút phù hợp với một phần tử trong pattern, thuật toán tiếp tục so sánh phần tử tiếp theo trong pattern với các nút con của nút hiện tại. Nếu không có nút con phù hợp, thuật toán sẽ quay lại và tìm kiếm trên các nhánh khác của cây.

Để đảm bảo tính chính xác, thuật toán:

* Duyệt qua từng nhánh của cây, tìm kiếm các phần tử của pattern.
* Kiểm tra tất cả các nút con để tìm sự trùng khớp với phần tử hiện tại của pattern.
* Nếu một nút trùng khớp với phần tử trong pattern, đệ quy tiếp tục tìm kiếm phần tử kế tiếp của pattern từ nút con này.

## 1.3. Các bước của thuật toán

Khởi tạo biến đếm count bằng 0, là số lượng lần xuất hiện của pattern trong cây.

Gọi hàm đệ quy TraverseAndCount, bắt đầu từ nút gốc của cây và phần tử đầu tiên trong pattern.

Trong hàm TraverseAndCount, thực hiện các bước sau:

* Kiểm tra xem patternIndex có đạt đến độ dài của pattern chưa:
  + Nếu có, điều này có nghĩa là tất cả phần tử của pattern đã được tìm thấy trong cây. Tăng count bằng số lượng của nút hiện tại (node.Count).
  + Nếu chưa, lấy phần tử hiện tại của pattern dựa trên patternIndex.
* Duyệt qua tất cả các nút con của nút hiện tại (node.Children.Values):
  + Kiểm tra xem ItemData của nút con có trùng với phần tử hiện tại của pattern không, bằng cách sử dụng hàm checkContains.
    - Nếu có, tiếp tục gọi đệ quy TraverseAndCount với patternIndex + 1 để tìm phần tử tiếp theo trong pattern trên nhánh hiện tại.
    - Nếu không, tiếp tục đệ quy tìm pattern trên nhánh hiện tại với patternIndex không đổi.

Sau khi hàm đệ quy hoàn tất, giá trị của count là số lượng lần pattern xuất hiện trong cây.

# 2. Triển khai thuật toán

Đoạn code dưới đây sẽ thêm vào trong file **Tree.cs**

| public int CountPattern(Node node, List<Item> pattern) {  int count = 0;   // Gọi hàm đệ quy phụ trợ để duyệt cây  TraverseAndCount(node, pattern, 0, ref count);   return count; }  // Hàm phụ trợ đệ quy để duyệt và đếm private void TraverseAndCount(Node node, List<Item> pattern, int patternIndex, ref int count) {  // Nếu đã tìm hết các phần tử trong pattern, tăng biến đếm  if (patternIndex == pattern.Count)  {  count += node.Count;  return;  }   // Lấy phần tử hiện tại trong pattern  string currentItemName = pattern[patternIndex].Name;   // Kiểm tra tất cả các nút con của node hiện tại  foreach (var child in node.Children.Values)  {  // Nếu tên của nút con trùng với tên phần tử hiện tại  if (child.ItemData != null)  {  //if (child.ItemData.Name == currentItemName)  if (checkContains(child.ItemData.Name, pattern))  {  // Đệ quy tiếp tục với nút con này và phần tử tiếp theo trong pattern  TraverseAndCount(child, pattern, patternIndex + 1, ref count);  }  else  {  //Tiếp tục tìm phần tử hiện tại của pattern trên nhánh hiện tại  TraverseAndCount(child, pattern, patternIndex, ref count);  }  }  } } private bool checkContains(string nameNode, List<Item> pattern) {  for (int i = 0; i < pattern.Count; i++)  {  if (pattern[i].Name == nameNode)  return true;  }  return false; } |
| --- |

# 3. Cập nhật một số hàm liên quan

## 3.1. Hàm tìm tập phổ biến

Ở phương pháp cũ ta lưu tập phổ biến dạng List<List<string>> rồi lại đếm lại tập phổ biến xuất hiện bao nhiêu lần khi tìm luật kết hợp. Thay vì như vậy, ta sẽ lưu dạng List<ItemSet>. ItemSet có các thuộc tính như label, support.

| public List<ItemSet> GetFrequentItemsets() {  var c1ItemList = GenerateC1(transaction);  var l1ItemList = PruneL1(c1ItemList);  SortItemsDescending(l1ItemList);   var fpTree = BuildFPTree(transaction, l1ItemList);  List<List<Node>> allBranches = new List<List<Node>>();   var pathListByItem = GetPathsFromTree(fpTree);  List<List<Item>> conditionalPatternBase = GetConditionalPatternBase(pathListByItem, l1ItemList);    var uniqueCombinations = new HashSet<string>();  var frequentItemsets = new List<ItemSet>();   for (int index = 0; index < conditionalPatternBase.Count; index++)  {  var itemList = conditionalPatternBase[index];  for (int k = 1; k <= itemList.Count; k++)  {  // Lấy tất cả các kết hợp có độ dài k của itemList  var combinations = Helper.GetCombinations(itemList, k);  foreach (var combination in combinations)  {  // Lấy danh sách tên của các item trong combination và sắp xếp để tạo chuỗi định danh duy nhất  var combinationNames = combination.Select(i => i.Name).OrderBy(name => name).ToList();  var combinationKey = string.Join(",", combinationNames);   // Nếu combinationKey chưa có trong uniqueCombinations  if (!uniqueCombinations.Contains(combinationKey))  {  uniqueCombinations.Add(combinationKey);   // Gọi hàm CountPattern để đếm số lần xuất hiện của combination trong cây FP-Tree  int count = fpTree.CountPattern(fpTree.Root, combination.ToList());   // Thêm combination vào frequentItemsets với số lần xuất hiện đã đếm được  frequentItemsets.Add(new ItemSet  {  Label = combinationKey,  Support = count  });  }  }  }  }  var sortedFrequentItemsets = frequentItemsets  .Where(itemSet => itemSet.Support >= minSupportCount)  .OrderBy(itemSet => itemSet.Label.Split(',').Length)  .ThenBy(itemSet => itemSet.Label)  .ToList();  return sortedFrequentItemsets; } |
| --- |

## 3.2. Hàm in tập phổ biến

| /// <summary>  /// In danh sách các tập phổ biến, nhóm theo độ dài và sắp xếp.  /// </summary>  public static void PrintFrequentItemsets(List<ItemSet> frequentItemsets, int minSup)  {  var groupedByLength = frequentItemsets  .GroupBy(itemset => itemset.Label.Split(',').Length)  .OrderBy(g => g.Key);   foreach (var group in groupedByLength)  {  Console.WriteLine($"\nFrequent ItemSet (Length = {group.Key}, Support >= {minSup}):");  foreach (var itemset in group.OrderBy(itemset => itemset.Label))  {  Console.WriteLine($"Item: {itemset.Label}, Count: {itemset.Support}");  }  }  } |
| --- |

## 3.3. Hàm sinh luật dựa trên tập phổ biến mới

| /// <summary>  /// Sinh tất cả các luật kết hợp từ danh sách tập phổ biến và bảng đếm.  /// </summary>  public static List<AssociationRule> GetAssociationRules(List<ItemSet> frequentItemsets, int totalTransactions, int minSup)  {  var countTable = frequentItemsets.ToDictionary(itemset => itemset.Label, itemset => itemset.Support);  var rules = new List<AssociationRule>();   foreach (var itemset in frequentItemsets)  {  var items = itemset.Label.Split(',').Select(i => i.Trim()).ToList();  double supportItemset = (double)itemset.Support / totalTransactions;   for (int k = 1; k < items.Count; k++)  {  foreach (var combination in GetCombinations(items, k))  {  var consequent = items.Except(combination).ToList();  if (!consequent.Any()) continue;  string combinationKey = string.Join(",", combination.OrderBy(x => x));  if (countTable.ContainsKey(combinationKey))  {  double supportCombination = (double)countTable[combinationKey] / totalTransactions;  double confidence = supportCombination == 0 ? 0 : Math.Round(supportItemset \* 100 / supportCombination, 2);  if (supportItemset >= (double)minSup / totalTransactions)  {  rules.Add(new AssociationRule  {  Label = $"{combinationKey} => {string.Join(", ", consequent)}",  Support = Math.Round(supportItemset \* 100, 2),  Confidence = confidence  });  }  }  else  {  Console.WriteLine($"Warning: Key '{combinationKey}' not found in countTable.");  }  }  }  }  return rules.OrderByDescending(r => r.Support).ThenByDescending(r => r.Confidence).ToList();  } |
| --- |

# 4. Kết quả thực hiện

Thuật toán FPGrowth

| **Số lượng bộ dữ liệu** | **Phương pháp cũ** | **Phương pháp tối ưu** |
| --- | --- | --- |
| Dữ liệu mẫu (minsup = 3) | 00:00:00.0644157 ms | 00:00:00.0604494 |
| 500 giao dịch (minsup = 3) | 00:00:00.0901262 ms | 00:00:00.0793138 |
| 1000 giao dịch (minsup = 4) | 00:00:00.4614085 ms | 00:00:00.1341678 |
| 1500 giao dịch (minsup = 5) | 00:00:00.6167843 ms | 00:00:00.1380063 |
| 2000 giao dịch (minsup = 6) | 00:00:00.9252857 ms | 00:00:00.1572310 |
| 3000 giao dịch (minsup = 8) | 00:00:01.1607066 ms | 00:00:00.1983381 |
| 5000 giao dịch (minsup = 12) | 00:00:01.6537924 ms | 00:00:00.2474607 |
| 7000 giao dịch (minsup = 16) | 00:00:02.2031810 ms | 00:00:00.2914873 |
| 10000 giao dịch (minsup = 22) | 00:00:02.1629009 ms | 00:00:00.3158778 |
| 15000 giao dịch (minsup = 32) | 00:00:03.8292306 ms | 00:00:00.5720914 |
| 20000 giao dịch (minsup = 42) | 00:00:04.5443467 ms | 00:00:00.5220304 |
| 30000 giao dịch (minsup = 62) | 00:00:09.0921478 ms | 00:00:00.7728635 |
| 50000 giao dịch (minsup = 102) | 00:00:11.8927231 ms | 00:00:01.4528304 |
| 100000 giao dịch (minsup = 202) | 00:00:47.7117791 ms | 00:00:02.8336607 |
| 500000 giao dịch (minsup = 1002) | 00:03:58.5601073 ms | 00:00:14.7906712 |
| 1000000 giao dịch (minsup = 2002) |  | 00:00:28.2675331 |