# **Câu 2: Dùng MapReduce trong Hadoop để tìm cặp item phổ biến**

## ***Tìm Baskets***

Với đầu vào là file baskets.csv, đầu tiên, chúng tôi cài đặt file FindingBaskets.java. Mapper của Finding Baskets job sẽ tạo key từ 2 trường Member\_number và Date và value là trường itemDescription.

Context sẽ viết xuống dưới dạng, ví dụ: 1247-22/01/2014 [\tab] whole milk. Lưu ý rằng key và value phân cách nhau bởi dấu tab.

Sau đó, Reducer đã nhận được mảng các item, lúc này, Reducer sẽ đưa mảng này vào HashSet để tránh item bị trùng lắp, tiếp tục dùng mảng đã xử lý trùng lắp chuyển thành ArrayList để cho việc sắp xếp các item theo bảng chữ cái (để thuận tiện cho việc sinh cặp item cho thuật toán Apriori) và tiếp tục dùng method toString() để chuyển thành dạng String, ví dụ, ta có mảng [soda, whole milk] nó sẽ chuyển thành dạng string “[soda, whole milk]”, tiếp tục thay thế khoảng cách sau dấu phẩy (,) và loại bỏ hai dấu ngoặc vuông ([ ]), ta được string sau: soda,whole milk. Sau đó, context viết xuống file với key là một Text có giá trị “” (string trống) và value là Text của các item phân cách nhau bởi dấu phẩy như ví dụ vừa rồi.

A screenshot of a computer screen

Description automatically generated

## ***Apriori Pass 1***

Đầu vào Apriori Pass 1 job là output file part-r-00000 của Finding Baskets job trên HDFS, tạm gọi file này là finding\_baskets, đồng thời còn nhận vào một biến tên là supportThreshold được cài đặt trong configuration và cũng là tham số truyền vào của người dùng khi biên dịch và chạy code, lưu ý rằng supportThreshold là một con số nguyên dương cụ thể, không phải một số thập phân như các section sau đề cập. Với mỗi dòng trong file này, Mapper lấy từng item và viết xuống dưới dạng [item] [\tab] [1], ví dụ: whole milk [\tab] 1.

Reducer của job này, với mỗi key nó tính tổng lại mảng giá trị 1, kiểm tra xem tổng này có lớn hơn hoặc bằng supportThreshold hay không, nếu thoả mãn thì giữ lại và ghi xuống file output dưới dạng [item] [\tab] [count của nó], ví dụ: whole milk [\tab] 26. File cho job này là AprioriPass1.java

A screenshot of a computer

Description automatically generated

## ***Apriori Pass 2***

Đầu vào của Apriori Pass 2 job là output file finding\_baskets của Finding Basket job, một Cache File là file output part-r-00000 của Apriori Pass 1 job, file này đóng vai trò là một broadcast file gửi đi cho từng MapTask, tạm gọi file này là frequent\_items và cuối cùng vẫn là supportTheshold như Pass 1.

Mapper của job, đầu tiên, đọc file frequent\_item để tạo một HashSet chứa các item phổ biến gọi biến lưu set này là frequentItems thông qua hàm setup. Hàm map tạo một HashSet chứa các item trong basket (mỗi dòng), lấy giao giữa nó và frequentItems. Tiếp tục sinh ra các cặp phổ biến từ phần giao đó. Cuối cùng, cho giá trị 1 cho mỗi cặp, context viết xuống dưới dạng [{item1, item2}] [\tab] [1], ví dụ: {bread, whole milk} [\tab] 1.

Reducer của job, tiếp tục tính tổng mảng giá trị 1 cho mỗi cặp, kiểm tra xem tổng đó có lớn hơn hoặc bằng supportThreshold hay không và ghi xuống output file các cặp thoả mãn dưới dạng [{item1, item2}] [\tab] [count của nó]. File cho job này là AprioriPass2.

A screenshot of a computer program

Description automatically generated

# **Câu 3: Cài đặt thuật toán PCY bằng PySpark**

Tạo class PCY, các tham số khởi tạo cần quan tâm gồm min\_support là một số thực (phân biệt với supportThreshold ở câu 2) và min\_confidence cũng là một số thực và num\_basket là một số nguyên dương. Phương thức fit nhận vào một DataFrame, nó sẽ tìm ra min\_count (tích của min\_support và số lượng basket).

## ***PCY Pass 1***

Sau đó, nó gọi phương thức pass1, phương thức pass1 tìm ra được item phổ biến cùng với count của nó, lưu nó với dạng dict với key là item (string) và value là count của nó (int). Tương tự, pass1 cũng tìm ra các bucket phổ biến lưu nó ở dạng set hay tập hợp các bucket phổ biến, chúng tôi lựa chọn lưu bucket phổ biến ở dạng set để giảm bộ nhớ đồng thời việc tìm kiếm chỉ mất thời gian O(1) ( nên gõ complexity ở dạng equation ). Về hàm hash cho các bucket, ở đây chúng tôi sử dụng một hàm hash riêng cho từng item và một hàm hash cho cặp item.

Hàm hash cho một item

Trong đó:

* là hàm chuyển ký tự thành giá trị ASCII
* là vị trí các kí tự có trong item (string)
* là ký tự thứ trong item

Hàm hash cho item này cố gắng hướng tới phân phối đều và nhấn mạnh vị trí của các item nhằm tránh việc các item khác nhau nhưng có kí tự giống nhau sẽ có giá trị hash như nhau.

Hàm hash cho cặp item

Hàm hash cho cặp item này đảm bảo các giá trị khi được hash chỉ rơi vào số lượng bucket. Cần lưu ý rằng việc hash cặp item ở basket vào bucket của chúng tôi khác với cách cài đặt thuật toán PCY gốc do chúng tôi đã tìm thấy các item phổ biến trước đó, nên chúng tôi chỉ quan tâm các cặp item phổ biến và hash chúng vào các bucket.

## ***PCY Pass 2***

Ở Pass 2, với mỗi basket, chúng tôi chỉ lấy các item phổ biến trong đó, sinh ra các cặp item từ nó, kiểm tra cặp item có trong hash table (các bucket) hay không, nếu thoả mãn, mỗi cặp được gán giá trị 1. Sau đó, gộp các cặp item lại và đếm, kiểm tra xem count của nó lớn hơn min\_count hay không, nếu thoả mãn thì giữa lại. Kết thúc quá trình này, chúng tôi thu được một dictionary có key là cặp item và value là count của nó đã thoả mãn điều kiện lớn hơn min\_count. Tạo DataFrame cho cặp item phổ biến và count của nó.

A white background with black text

Description automatically generated

## ***Association rules***

Với mỗi cặp item phổ biến vừa tìm được, tận dụng việc lưu các item phổ biết và count của nó, cặp item phổ biến và count của nó đều ở dạng dictionary, chúng tôi tìm luật liên kết bằng cách tính cho 2 trường hợp và . Kiểm tra xem cái nào lớn hơn hoặc bằng min\_confident thì giữ lại. Tạo DataFrame cho association rule cho từng cặp phổ biến thoả mãn.

A screenshot of a computer

Description automatically generated

# **Câu 4: Cài đặt in-memory cho Apriori và PCY, đo thời gian thực hiện của chúng với kích thước dataset tăng dần**

Ở phần này chúng tôi vẫn tiếp tục cài đặt 2 class Apriori và PCY như mô tả ở các section trước. Tuy nhiên, chúng tôi đã cài đặt PCY đúng như bản gốc, tức là ở Pass 1, chúng tôi đồng thời đếm từng item, hash từng cặp item (đã sort) và các bucket. Sau đó lọc lấy các item phổ biến, bucket phổ biến. Và tiếp tục làm Pass 2 PCY như các section trước đã mô tả. Sau đó chúng tôi cài đặt một hàm để đo thời gian chạy của 2 thuật toán với kích thước dataset tăng dần, kết quả đo đạc bên dưới.

A graph with blue and orange lines

Description automatically generated