**COMP 4331**

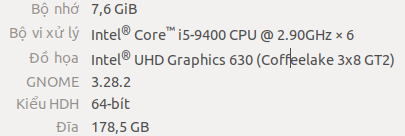
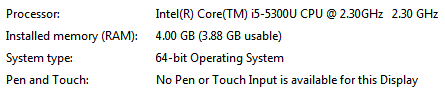
**Assignment 1 Report**

**PHAM Trung Kien**

**20553388**

**The environment:**

I have tried to run the programs in two different environments:

1. 
2. 

All the programs were implemented using Python 3. All the external libraries used for implementation has already been stated in the programs

**Experimental Result:**

All three implementation give the result of 699 frequent itemsets. Please go to the link below to see the result:

<https://drive.google.com/file/d/1mzQTuq-RU6ag77hDw6q3atChlG-OfKKX/view?usp=sharing>

Notice: The first column is the frequent itemsets

The second column is the respective frequency

**Part 2.1:**

The runtime for each program in respective environments are:

1. Apriori:

Environment 1: 1504.8992986679077

Environment 2: 2630.973483324051

1. Hash Tree:

Environment 1: 475.50319170951843

Environment 2: 760.8395178318024

**Part 2.2:**

The maximal frequent itemsets and the closed frequent itemsets are shown in these links below:

1. Maximal:

<https://drive.google.com/file/d/1yUo-sRvCK55XnHZja7wCQR2YQdf72WAe/view?usp=sharing>

1. Closed:

<https://drive.google.com/file/d/16UBaHGzDvgnaM6gm7MXsC_k1n5KyUZmL/view?usp=sharing>

Notice: The first column is the frequent itemsets

The second column is the respective frequency

**Reasons why their performances vary in term of efficiency:**

According to the runtime in both two environments, we can clearly see that the implementation of Apriori algorithm alone without making use of Hash Tree or FP Tree resulted in the longest runtime. With the use of Hash Tree, the Apriori algorithm implementation is executed a lot faster (nearly 4 times). Furthermore, implementing the FP Growth algorithm using FP Tree is the fastest one. They are due to:

For Apriori Algorithm (if implemented without the help of Hash Tree or FP Tree):

* It costs Apriori Algorithm (if implemented without the help of Hash Tree or FP Tree) a lot of resources to generating the candidates of specific length in such a large database
* It is tedious to scan through the database each time for counting frequency for each candidate (which is run extremely long when the list of candidates generated is huge)

For Apriori Implementation with Hash Tree:

* It still costs the implementation quite a lot of resources to generating the candidates of specific length in such a large database
* However, with the help of Hash Tree, after generating the candidates, we can insert them into the tree to provide much faster search for frequency with respect to the database by generating the subsets in each transaction and performing incrementing to the node of the tree using hash function. This helps reduce the amount of time spending for scanning through the database. However, this will not help in case of the amount of candidates generated is too small (Please find the explanation in the code)

(Extra)

* Finally, in case of FP Growth algorithm implemented using FP Tree, it is the best implementation among the three as it only need us to scan through the database just twice. In addition, the algorithm does not require us to generate the list of candidates, which help reduce a lot of resource spending on it like in the first two implementation. And also, the method helps to transform the database from such a huge file of data into a smaller compressed form, FP Tree, and perform searching by using divide-and-conquer method to get the frequency of the frequent itemsets.