Assignment 2: Frequent Itemsets

This assignment contains three problems. First, you will implement two algorithms to find frequent itemsets namely the Multi-Hash and Toivonen algorithms. You will also implement a program to use Spark for finding frequent pairs. The transactions (basket files) will be given as an input file for task 1 and 2. And the input files for task 3 are in the folder named Spark. The instructions about how to use the input files will be explained in details in each task section. The goal of the assignment is to make you understand the algorithms better by coding them.

Write your own code!

For this assignment to be an effective learning experience, you must write your own code! I emphasize this point because you will be able to find Python implementations of most or perhaps even all of the required functions on the web. Please do not look for or at any such code! **Do not share code with other students in the class!!**

Here's why:

- The most obvious reason is that it will be a huge temptation to cheat: if you include code written by anyone else in your solution to the assignment, you will be cheating. As mentioned in the syllabus, this is a very serious offense, and may lead to you failing the class.
- However, even if you do not directly include any code you look at in your solution, it surely will influence your coding. Put another way, it will short-circuit the process of you figuring out how to solve the problem, and will thus decrease how much you learn.

So, just don't look on the web for any code relevant to these problems. Don't do it.

Submission Details

For problems 1 and 2, you will turn in a Python program. For problem 3, you have the choice to turn in a Python program or a Scala program. This assignment accounts for 9% of the final class grade, and each program in this assignment accounts for 3% of the final class grade. If you choose to use Scala for your problem 3, you will receive a 50% bonus for your problem 3. For example, if your submission receives full credits for all three problems and you use Scala for problem 3, you will receive 3% + 3% + 4.5%, which is a total of 10.5 points for your assignment 2 towards your class grade. if your submissions receive half credits for all three problems and you use Scala for problem 3, you will receive 1.5% + 1.5% + 2.25%, which is a total of 5.25 points for your assignment 2 towards your class grade.

Your program should be able to read data in the format as provided in the Data folder. You can use the sample data (in the Data folder) and the solutions (in the Solution folder) to test your program, but we might test your program with other input data in the same format.

For this assignment, you need to turn in the following Python files.

1. A Python program that implements Multi-Hash : <firstname>_<lastname>_multihash.py
2. A Python program that implements Toivonen: <firstname>_<lastname>_toivonen.py
3. A Python or Scala program that runs on Spark: <firstname>_<lastname>_spark.py or <firstname>_<lastname>_spark.jar with <firstname>_<lastname>_spark.scala

Problem 1: Multi-Hash Algorithm (3% of the class grade)

Implement the multi-hash algorithm to generate ALL frequent itemsets: You need to use two independent hash functions in your program. Both hash functions should have the same number of buckets. In your output, you need to show the number of itemsets hashed into individual buckets. For example, the sample below uses two hash functions, each has three buckets where the 0:0 means there was no itemset hashed into bucket 0, and the 1:2 means two itemsets were hashed into bucket 1. Note that the counts in individual buckets vary depending on your hash function. You do not need to match the counts with the ones in the sample output files.

For example:

['a', 'b', 'c']

{0:0, 1:2, 3:5}
{0:1, 1:4, 3:2}
[['a', 'b']]

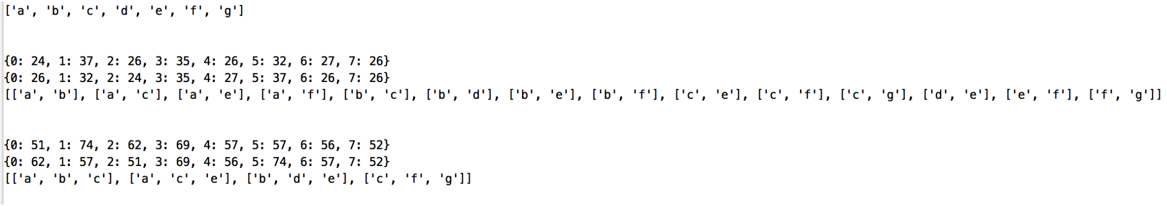
File Name: Please name your python script as <firstname>_<lastname>_multihash.py.

Executing code: python weiwei_duan_multihash.py input.txt 4 5

Where support = 4 and bucket size = 5

(Please confirm the output with output_multihash.txt)

The solution to input.txt is like the following snapshot.



```
['a', 'b', 'c', 'd', 'e', 'f', 'g']


{0: 24, 1: 37, 2: 26, 3: 35, 4: 26, 5: 32, 6: 27, 7: 26}
{0: 26, 1: 32, 2: 24, 3: 35, 4: 27, 5: 37, 6: 26, 7: 26}
[['a', 'b'], ['a', 'c'], ['a', 'e'], ['a', 'f'], ['b', 'c'], ['b', 'd'], ['b', 'e'], ['b', 'f'], ['c', 'e'], ['c', 'f'], ['c', 'g'], ['d', 'e'], ['e', 'f'], ['f', 'g']]


{0: 51, 1: 74, 2: 62, 3: 69, 4: 57, 5: 57, 6: 56, 7: 52}
{0: 62, 1: 57, 2: 51, 3: 69, 4: 56, 5: 74, 6: 57, 7: 52}
[['a', 'b', 'c'], ['a', 'c', 'e'], ['b', 'd', 'e'], ['c', 'f', 'g']]
```

Problem 2: Toivonen Algorithm (3% of the class grade)

Implement the Toivonen algorithm to generate ALL frequent itemsets: For this algorithm, you need to use a sample size of less than 60% of your entire dataset. You need to use an appropriate sampling method to obtain the random sample set. Then you need to perform a simple Apriori algorithm on the random samples to detect the frequent itemsets and the negative border. Your algorithm will finish to identify all frequent itemset once no itemsets in the negative borders have a frequency larger or equal to the support. If one or more itemsets in the negative borders have a frequency larger or equal to the support, your algorithm needs to resample the data and run the entire process again. You should use the slides and textbook as your guide to implement the Toivonen algorithm.

Input Parameters:

1. Input.txt: This is the input file containing all transactions. Each line corresponds to a transaction. Each transaction has items that are comma separated. Use input1.txt to test this algorithm.
2. Support: Integer that defines the minimum count to qualify as a frequent itemset.

Output:

Line 1 <number of iterations performed>

Line 2 <fraction of transactions used>

Line 3 onwards <frequent itemsets lexicographically sorted>

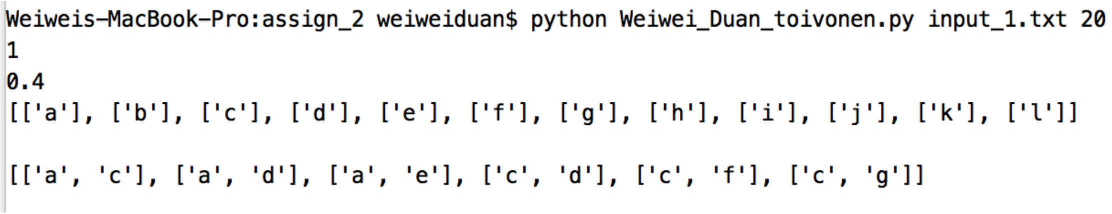
File Name: Please name your Python script as <firstname>_<lastname>_toivonen.py.

Executing code: **python weiwei_duan_toivonen.py input1.txt 20**

Where support = 20

(Please confirm the output with output_toivonen.txt)

The solution to input1.txt is like the following snapshot.



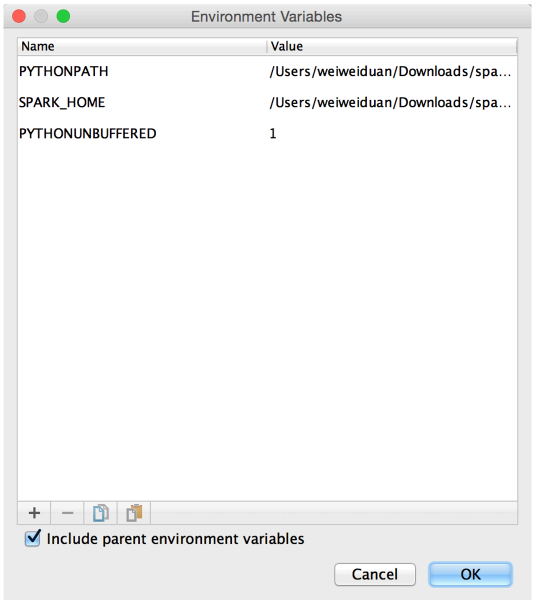
```
Weiweis-MacBook-Pro:assign_2 weiweiduan$ python Weiwei_Duan_toivonen.py input_1.txt 20
1
0.4
[['a'], ['b'], ['c'], ['d'], ['e'], ['f'], ['g'], ['h'], ['i'], ['j'], ['k'], ['l']]

[['a', 'c'], ['a', 'd'], ['a', 'e'], ['c', 'd'], ['c', 'f'], ['c', 'g']]
```

Problem 3: Finding frequent pairs on Spark (3% of the class grade)

You can write your solution to this problem in Scala or Python. The goal for this task is to find the frequent pairs among actresses and directors in the provided IMDB dataset. The raw data was downloaded from http://www.imdb.com/interfaces. You should use the data provided in the Spark folder and do not need to download the data again because we have cleaned the IMDB data for you. The cleaned sample data for this problem is in the folder named Spark. Each line in the sample data contains two elements. The first element is the name of the film. The second is the name of the actress or director. You should join the two datasets by using the name of the film. After the join, you should count the number of occurrence of each actress pair, director pair, and actress and director pair. Each line in output file should be a tuple like ((actress's name, director's name), frequency). For the solution provided with this assignment, the support was 5.

Tips on Running Your Code with the Python interpreter:

You need to add the paths to your Spark (path/to/your/Spark) and Python (path/to/your/Spark/python) folders to the interpreter environment variables named SPARK_HOME and PYTHONPATH, respectively. An example is shown in the following snapshot. After this step, you should add *from* *pyspark import SparkContext* in your codes. After that you can run and debug your codes using the Python interpreter.



Name	Value
PYTHONPATH	/Users/weiweiduan/Downloads/spa...
SPARK_HOME	/Users/weiweiduan/Downloads/spa...
PYTHONUNBUFFERED	1

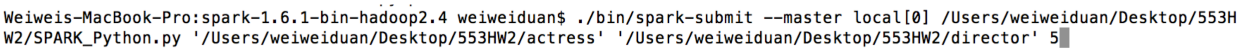
☑ Include parent environment variables

Tips on Running Your Code with Spark:

To run your code on Spark, we will use the spark-submit command in the bin folder (similar to what we did in the last assignment).

Input parameters:
1. actress: Each line in the file is a 2-tuple. The first element is the name of the film, and the second element is the name of the actress.
2. director: Each line in the file is a 2-tuple. The first element is the name of the film, and the second element is the name of the director.
3. Support: An integer number that defines a frequent itemset

An example is shown in the following snapshot.

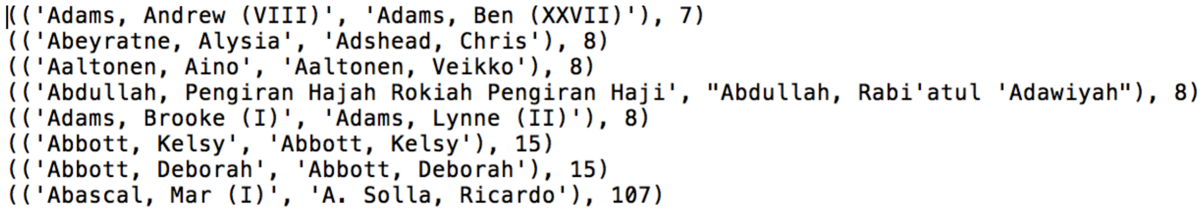


```
Weiweis-MacBook-Pro:spark-1.6.1-bin-hadoop2.4 weiweiduan$ ./bin/spark-submit --master local[0] /Users/weiweiduan/Desktop/553H
W2/SPARK_Python.py '/Users/weiweiduan/Desktop/553HW2/actress' '/Users/weiweiduan/Desktop/553HW2/director' 5
```

In the example, Support = 5

Output:
The result file named <firstname>_<lastname>_spark. Each line in the file should be a 2-tuple like this ((actress's name, director's name), frequency). The order of lines is by the frequency in the ascending order. Make sure you only produce one file. To do this, you can set the number of partition as 1.

The following snapshot is the outcome of the input example above.



```
(('Adams, Andrew (VIII)', 'Adams, Ben (XXVII)'), 7)
(('Abeyratne, Alysia', 'Adshead, Chris'), 8)
(('Aaltonen, Aino', 'Aaltonen, Veikko'), 8)
(('Abdullah, Pengiran Hajah Rokiah Pengiran Haji', "Abdullah, Rabi'atul 'Adawiyah"), 8)
(('Adams, Brooke (I)', 'Adams, Lynne (II)'), 8)
(('Abbott, Kelsy', 'Abbott, Kelsy'), 15)
(('Abbott, Deborah', 'Abbott, Deborah'), 15)
(('Abascal, Mar (I)', 'A. Solla, Ricardo'), 107)
```

Tips on Running Your Scala Code on Spark

You can refer the instruction in the first assignment about how to run Scala codes on Spark.

Input parameters: (Same as python one)
1. actress: Each line in the file is a 2-tuple. The first element is the name of the film, and the second element is the name of the actress.
2. director: Each line in the file is a 2-tuple. The first element is the name of the film, and the second element is the name of the director.
3. Support: An integer number that defines a frequent itemset

Output: (Same as python one)
The output should only be one file named <firstname>_<lastname>_spark. Each line in the file should be a 2-tuple like this: ((actress's name, director's name), frequency). The order of lines is by the frequency in the ascending order.

General Instructions:

1. Make sure your code compiles before submitting
2. Make sure to follow the output format and the naming format.
3. Make sure not to write the output to any files. Use standard output to print them.
4. We will be using Moss for plagiarism detection.

About Discussion Board on BlackBoard

In general, the questions posted on the Discussion Board will be answered within 24 hours on weekdays. During the weekends or holidays, we will answer the question on the next business day. Please make sure you start the assignment early so we can help you.