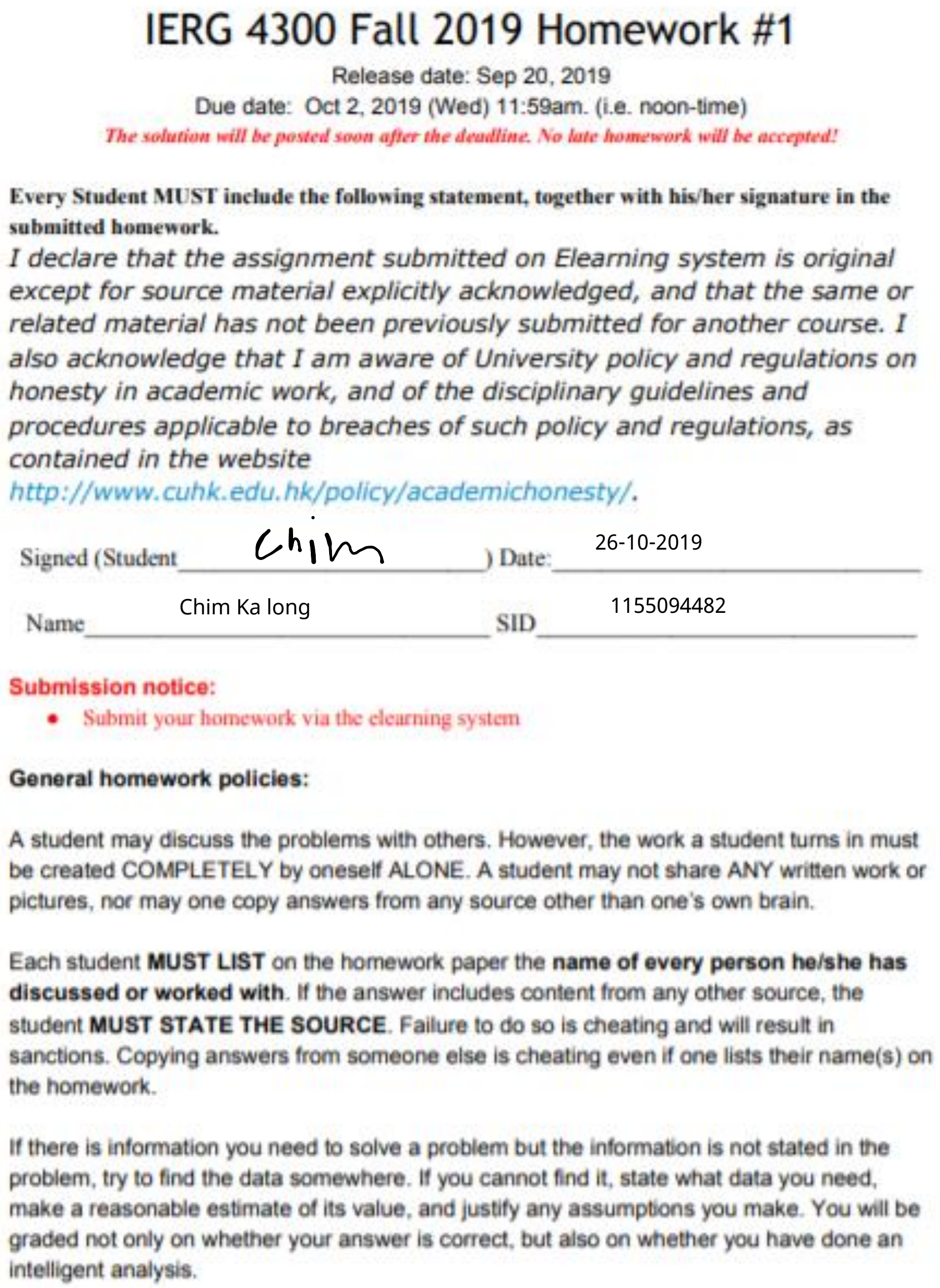
Chim Ka Long
1155094482



IERG 4300 Fall 2019 Homework #1

Release date: Sep 20, 2019

Due date: Oct 2, 2019 (Wed) 11:59am. (i.e. noon-time)

The solution will be posted soon after the deadline. No late homework will be accepted!

Every Student MUST include the following statement, together with his/her signature in the submitted homework.

I declare that the assignment submitted on Elearning system is original except for source material explicitly acknowledged, and that the same or related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website http://www.cuhk.edu.hk/policy/academichonesty/.

Signed (Student Chim) Date: 26-10-2019

Name Chim Ka long SID 1155094482

Submission notice:

- Submit your homework via the elearning system

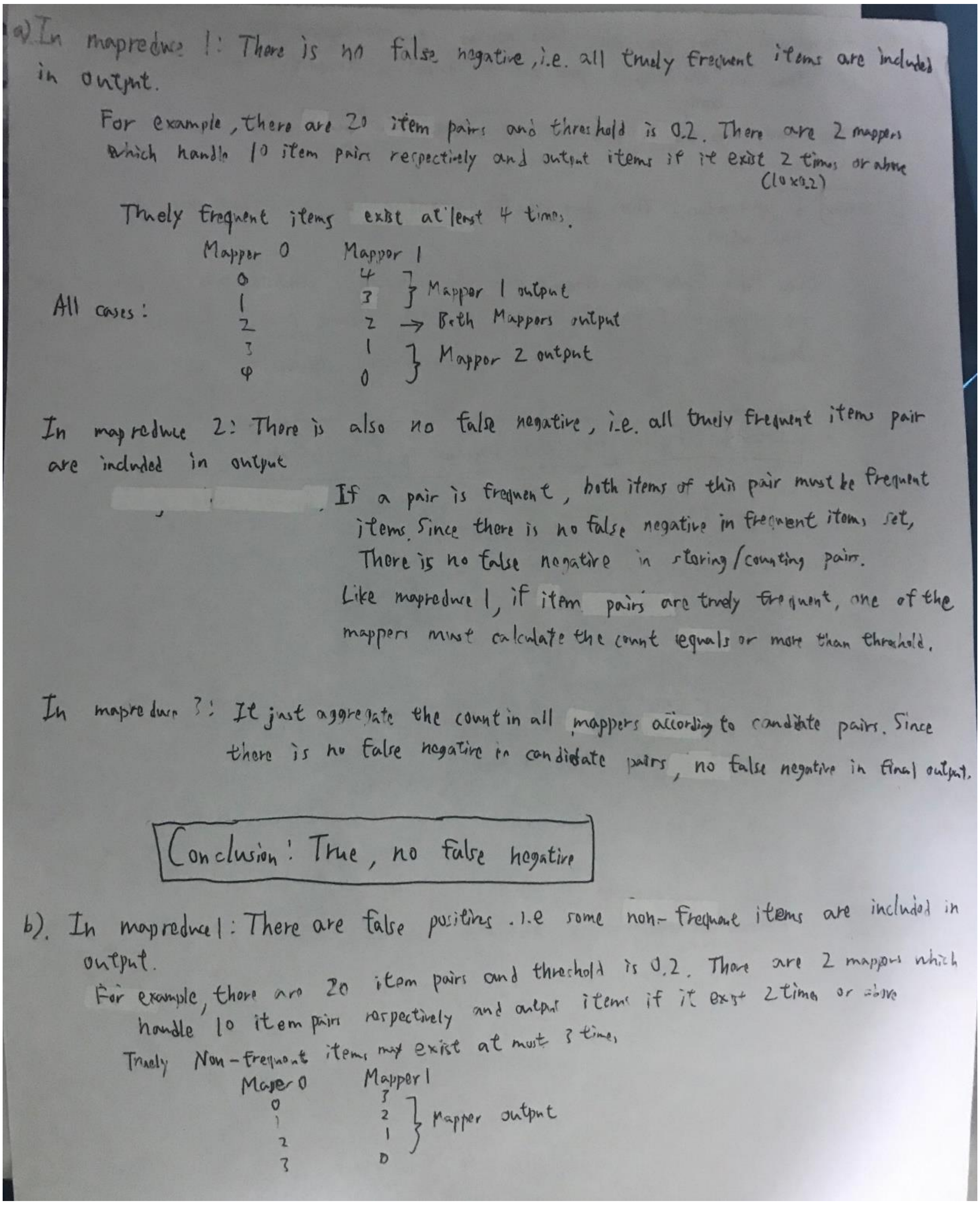
General homework policies:

A student may discuss the problems with others. However, the work a student turns in must be created COMPLETELY by oneself ALONE. A student may not share ANY written work or pictures, nor may one copy answers from any source other than one's own brain.

Each student **MUST LIST** on the homework paper the **name of every person he/she has discussed or worked with**. If the answer includes content from any other source, the student **MUST STATE THE SOURCE**. Failure to do so is cheating and will result in sanctions. Copying answers from someone else is cheating even if one lists their name(s) on the homework.

If there is information you need to solve a problem but the information is not stated in the problem, try to find the data somewhere. If you cannot find it, state what data you need, make a reasonable estimate of its value, and justify any assumptions you make. You will be graded not only on whether your answer is correct, but also on whether you have done an intelligent analysis.

Q0:



a) In mapredwe 1: There is no false negative, i.e. all truely frequent items are included in output.

For example, there are 20 item pairs and threshold is 0.2. There are 2 mappers which handle 10 item pairs respectively and output items if it exist 2 times or above (10×0.2)

Truely frequent items exist at least 4 times.

All cases:

Mapper 0	Mapper 1	
0	4	} Mapper 1 output
1	3	
2	2	→ Both Mappors output
3	1	} Mappor 2 output
4	0	

In mapredwe 2: There is also no false negative, i.e. all truely frequent items pair are included in output

If a pair is frequent, both items of this pair must be frequent items. Since there is no false negative in frequent items set, There is no false negative in storing/counting pair.

Like mapredwe 1, if item pairs are truely frequent, one of the mappers must calculate the count equals or more than threshold.

In mapredure 3: It just aggregate the count in all mappers according to candidate pairs. Since there is no false negative in candidate pairs, no false negative in final output.

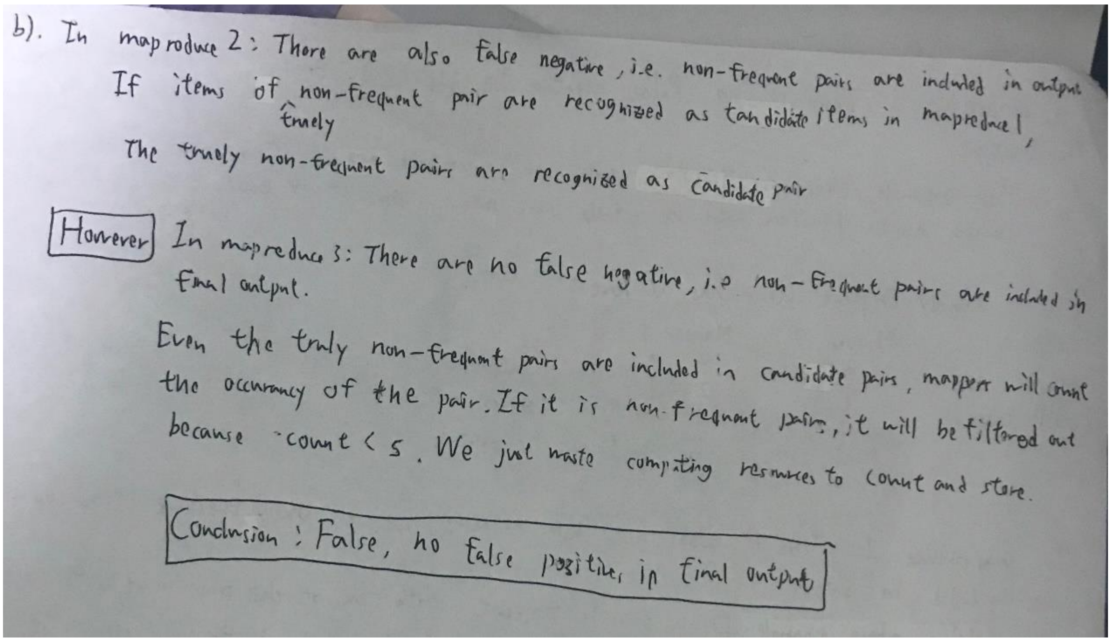
Conclusion: True, no false negative

b). In mapreduce 1: There are false positives. i.e some non-frequent items are included in output.

For example, there are 20 item pairs and threshold is 0.2. There are 2 mappers which handle 10 item pairs respectively and output items if it exist 2 times or above

Truely Non-frequent items may exist at most 3 times,

Maper 0	Mapper 1	
0	3	} Mapper output
1	2	
2	1	
3	0	



b). In maproduce 2: There are also false negative, i.e. non-frequent pairs are included in output

If items of truely non-frequent pair are recognized as candidate items in mapreduce 1,

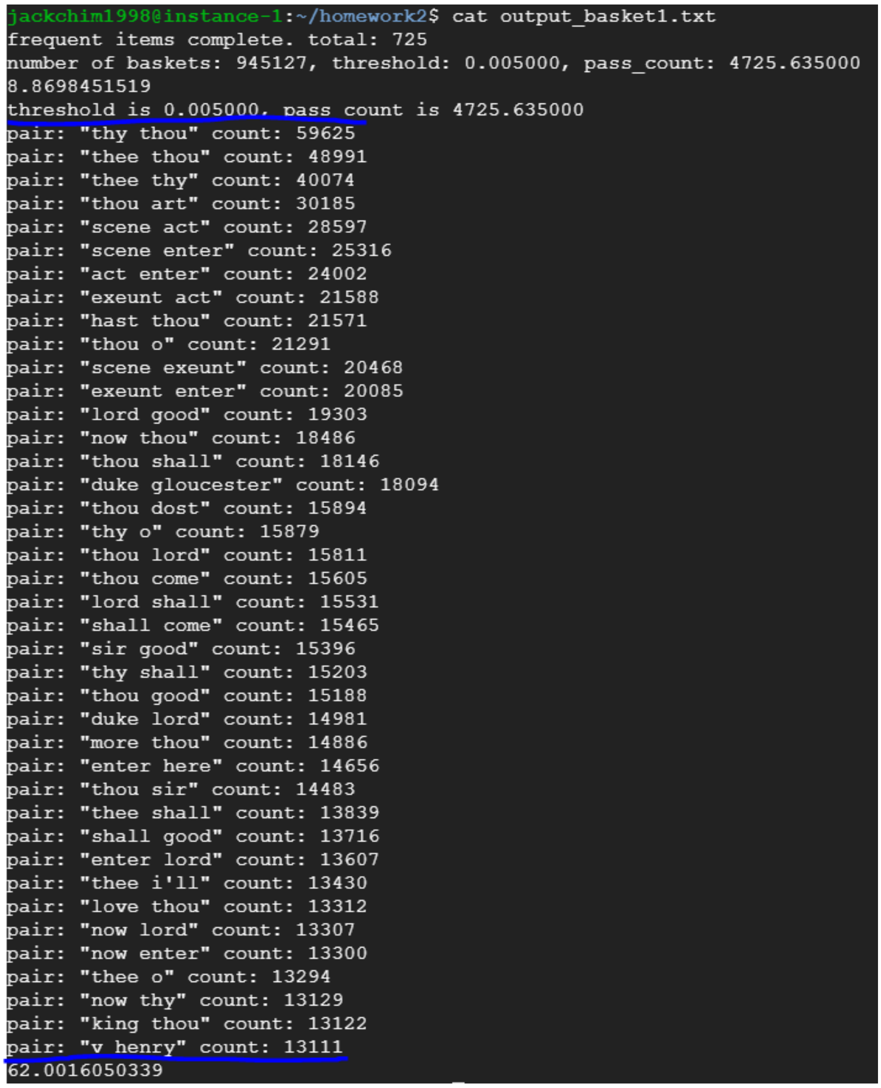
The truely non-frequent pairs are recognized as candidate pair

However In mapreduce 3: There are no false negative, i.e non-frequent pairs are included in final output.

Even the truly non-frequent pairs are included in candidate pairs, mapper will count the occurrancy of the pair. If it is non-frequent pair, it will be filtered out because count < 5. We just waste computing resources to count and store.

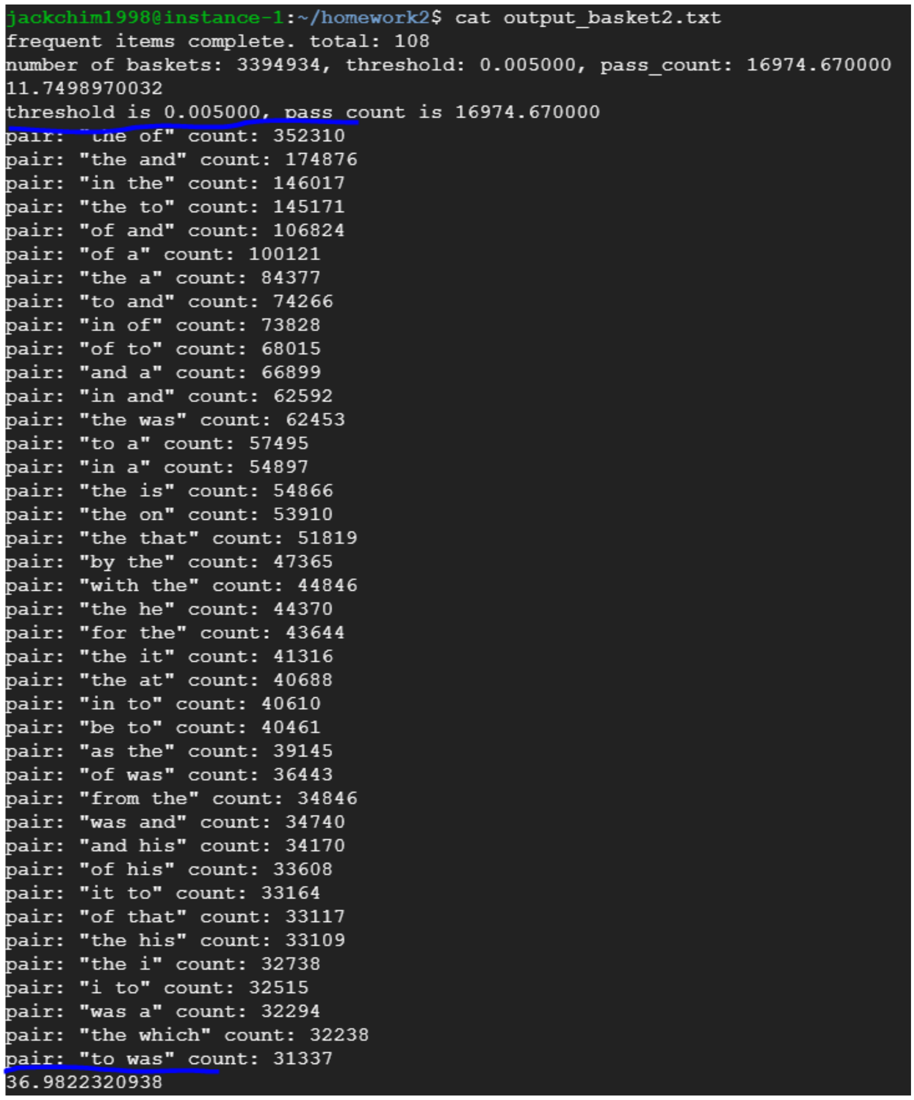
Conclusion: False, no false positive in final output

Q1a) basket1 result:



```
jackchim1998@instance-1:~/homework2$ cat output_basket1.txt
frequent items complete. total: 725
number of baskets: 945127, threshold: 0.005000, pass_count: 4725.635000
8.8698451519
threshold is 0.005000, pass count is 4725.635000
pair: "thy thou" count: 59625
pair: "thee thou" count: 48991
pair: "thee thy" count: 40074
pair: "thou art" count: 30185
pair: "scene act" count: 28597
pair: "scene enter" count: 25316
pair: "act enter" count: 24002
pair: "exeunt act" count: 21588
pair: "hast thou" count: 21571
pair: "thou o" count: 21291
pair: "scene exeunt" count: 20468
pair: "exeunt enter" count: 20085
pair: "lord good" count: 19303
pair: "now thou" count: 18486
pair: "thou shall" count: 18146
pair: "duke gloucester" count: 18094
pair: "thou dost" count: 15894
pair: "thy o" count: 15879
pair: "thou lord" count: 15811
pair: "thou come" count: 15605
pair: "lord shall" count: 15531
pair: "shall come" count: 15465
pair: "sir good" count: 15396
pair: "thy shall" count: 15203
pair: "thou good" count: 15188
pair: "duke lord" count: 14981
pair: "more thou" count: 14886
pair: "enter here" count: 14656
pair: "thou sir" count: 14483
pair: "thee shall" count: 13839
pair: "shall good" count: 13716
pair: "enter lord" count: 13607
pair: "thee i'll" count: 13430
pair: "love thou" count: 13312
pair: "now lord" count: 13307
pair: "now enter" count: 13300
pair: "thee o" count: 13294
pair: "now thy" count: 13129
pair: "king thou" count: 13122
pair: "v henry" count: 13111
62.0016050339
```

Basket2 result:



```
jackchim1998@instance-1:~/homework2$ cat output_basket2.txt
frequent items complete. total: 108
number of baskets: 3394934, threshold: 0.005000, pass_count: 16974.670000
11.7498970032
threshold is 0.005000, pass count is 16974.670000
pair: "the of" count: 352310
pair: "the and" count: 174876
pair: "in the" count: 146017
pair: "the to" count: 145171
pair: "of and" count: 106824
pair: "of a" count: 100121
pair: "the a" count: 84377
pair: "to and" count: 74266
pair: "in of" count: 73828
pair: "of to" count: 68015
pair: "and a" count: 66899
pair: "in and" count: 62592
pair: "the was" count: 62453
pair: "to a" count: 57495
pair: "in a" count: 54897
pair: "the is" count: 54866
pair: "the on" count: 53910
pair: "the that" count: 51819
pair: "by the" count: 47365
pair: "with the" count: 44846
pair: "the he" count: 44370
pair: "for the" count: 43644
pair: "the it" count: 41316
pair: "the at" count: 40688
pair: "in to" count: 40610
pair: "be to" count: 40461
pair: "as the" count: 39145
pair: "of was" count: 36443
pair: "from the" count: 34846
pair: "was and" count: 34740
pair: "and his" count: 34170
pair: "of his" count: 33608
pair: "it to" count: 33164
pair: "of that" count: 33117
pair: "the his" count: 33109
pair: "the i" count: 32738
pair: "i to" count: 32515
pair: "was a" count: 32294
pair: "the which" count: 32238
pair: "to was" count: 31337
36.9822320938
```

Remark: the number in last line is the elapsed time

Algorithm:

The words in blanket is the variable name in code or explanation.

In pass 1, split each line of input into list of word and process (filter duplicate words) them. Create dictionary (dic) with “word” as key and “count of word” as value. Then loop over whole input file and count the number of baskets (no_of_baskets).

After the pass 1, we can calculate the minimum number of count (pass_count) to pass the threshold (threshod * no_of_baskets). Loop over the dictionary(dic) and put the frequent words into frequent dictionary(dic_freq) which is consist of “word” as key and “code” as value. Then create a 2D array (pair_matrix) which has length and width same as number of frequent words to store number of pair of frequent words. We use 2D array instead of triangular matrix because it is easy to implement, and the memory consumption is enough for 2 sample input.

In pass 2, loop over the input and count the pair of items in same basket into 2d array if the both items of pair are frequent items.

After pass 2, we create a list(code_list) with code as index and word as element to let us find word quickly by code. Also, we create array(pair_arr_freq) to store the pair and count. Then loop over 2D array(pair_matrix) and sum up count (index i,j) and count (index j,i) into pair class (x,y,count)(x and y are code not word) which will be stored in pair_arr_freq. Finally, we sort the pair_arr_freq according to count in descending order and cut the end those elements do not pass the threshold. Print out the pair in words according to code_list and count.

Q1b) The result is same as 1a.

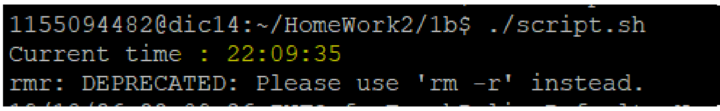
Algorithm:

The mapreduce job0 find candidate pair which is still possible to be frequent pairs. Mapper0 use the A-priori algorithm which in same as 1a. The key idea is a frequent pair must pass the number (local number of baskets in each mapper * threshold) in one of mappers. Reducer0 just output the pair to file local_can_pair_basket*.txt.

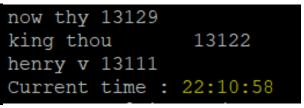
The mapreduce job1 count all pairs if they are candidate pair and filter out those pairs which do not satisfy with threshold. That is why we use local_can_pair_basket*.txt as supplementary file to mapper1. In reducer1, we need to know the total number of baskets to calculate pass line (threshold * number of baskets), so that we need count.txt as supplementary file to reducer1.

Comparison between 1a and 1b:

With basket1 as input, 1a use 62s and 1b use 93s.

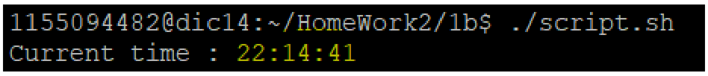


```
1155094482@dic14:~/HomeWork2/1b$ ./script.sh
Current time : 22:09:35
rmr: DEPRECATED: Please use 'rm -r' instead.
```

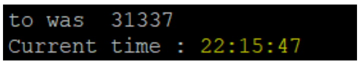


```
now thy 13129
king thou       13122
henry v 13111
Current time : 22:10:58
```

With basket2 as input, 1a use 37s and 1b use 66s.



```
1155094482@dic14:~/HomeWork2/1b$ ./script.sh
Current time : 22:14:41
```



```
to was  31337
Current time : 22:15:47
```

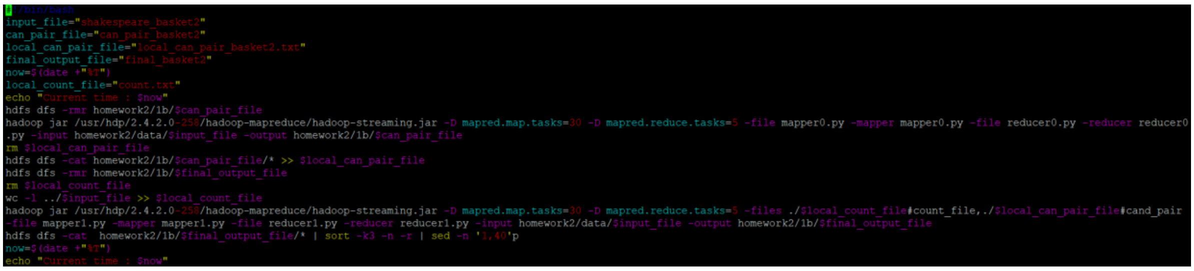
Why using mapreduce consume more time than running in single machine?

First, we need to see the script first. In script, we have command to delete directory of previous output file in HDFS and command to get the result of mapreduce job0 down. Those action will cause overhead.

Second, the mapreduce specific 30 mappers and 5 reducers. These number affect the performance. Also, others may also run job on these machines, so that we may need to wait for them finish. However, that is not my situation.

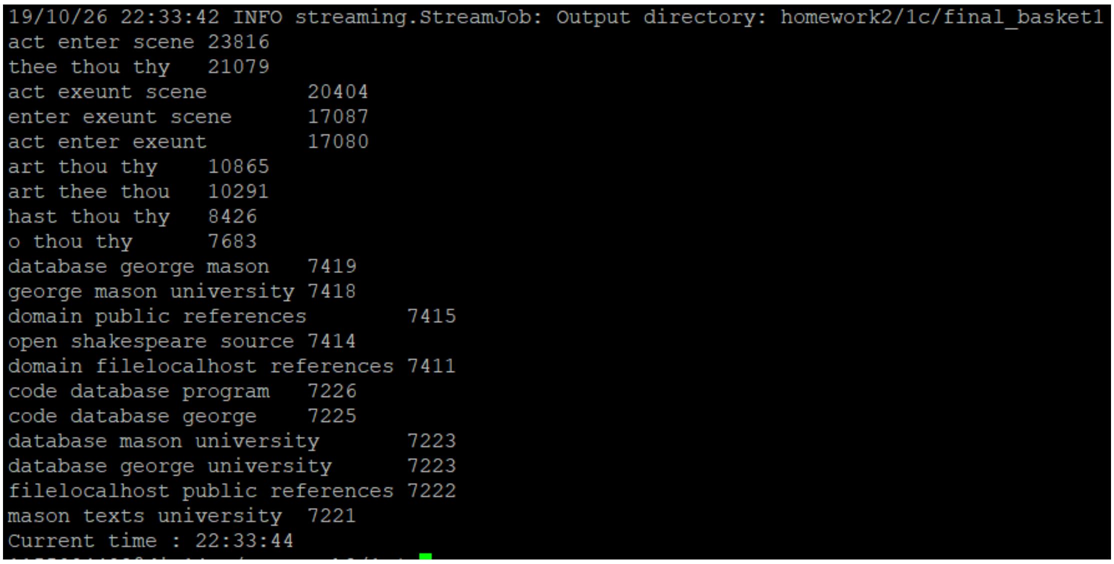
Third, the input file is transferred between machines. Network performance is affected by network overhead. Even the file is very small, we still need to pay those time of overhead.

As a result, If the input file is much larger e.g. 100GB, the overhead may become smaller in percentage. If we use more mappers and reducers, the time of calculating can also be smaller.



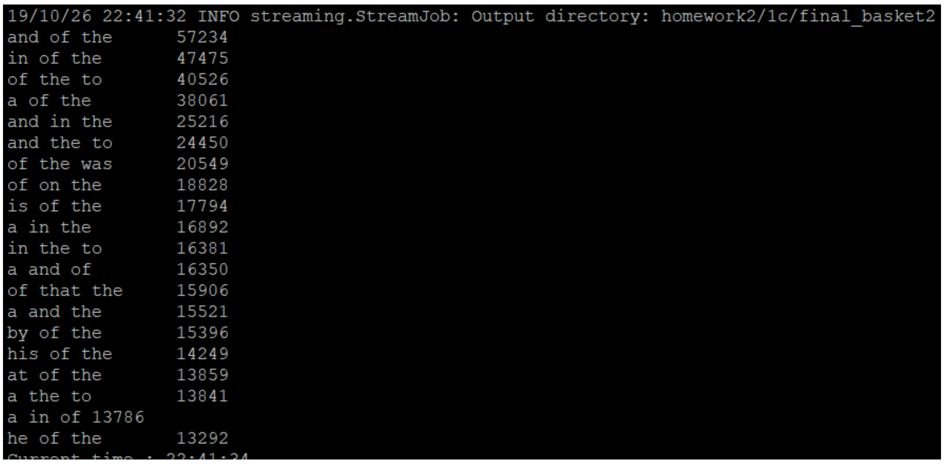
```
#!/bin/bash
input_file="shakespeare_basket2"
can_pair_file="can_pair_basket2"
local_can_pair_file="local_can_pair_basket2.txt"
final_output_file="final_basket2"
now=$(date +"%T")
local_count_file="count.txt"
echo "Current time : $now"
hdfs dfs -rmr homework2/1b/$can_pair_file
hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-streaming.jar -D mapred.map.tasks=30 -D mapred.reduce.tasks=5 -file mapper0.py -mapper mapper0.py -file reducer0.py -reducer reducer0
.py -input homework2/data/$input_file -output homework2/1b/$can_pair_file
rm $local_can_pair_file
hdfs dfs -cat homework2/1b/$can_pair_file/* >> $local_can_pair_file
hdfs dfs -rmr homework2/1b/$final_output_file
rm $local_count_file
wc -l ../$input_file >> $local_count_file
hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-streaming.jar -D mapred.map.tasks=30 -D mapred.reduce.tasks=5 -files ./$local_count_file#count_file,./$local_can_pair_file#cand_pair
-file mapper1.py -mapper mapper1.py -file reducer1.py -reducer reducer1.py -input homework2/data/$input_file -output homework2/1b/$final_output_file
hdfs dfs -cat  homework2/1b/$final_output_file/* | sort -k3 -n -r | sed -n '1,40'p
now=$(date +"%T")
echo "Current time : $now"
```

Q1c) result of basket1:



```
19/10/26 22:33:42 INFO streaming.StreamJob: Output directory: homework2/1c/final_basket1
act enter scene 23816
thee thou thy   21079
act exeunt scene        20404
enter exeunt scene      17087
act enter exeunt        17080
art thou thy    10865
art thee thou   10291
hast thou thy   8426
o thou thy      7683
database george mason   7419
george mason university 7418
domain public references        7415
open shakespeare source 7414
domain filelocalhost references 7411
code database program   7226
code database george    7225
database mason university       7223
database george university      7223
filelocalhost public references 7222
mason texts university  7221
Current time : 22:33:44
```

Result of basket2:



```
19/10/26 22:41:32 INFO streaming.StreamJob: Output directory: homework2/1c/final_basket2
and of the      57234
in of the       47475
of the to       40526
a of the        38061
and in the      25216
and the to      24450
of the was      20549
of on the       18828
is of the       17794
a in the        16892
in the to       16381
a and of        16350
of that the     15906
a and the       15521
by of the       15396
his of the      14249
at of the       13859
a the to        13841
a in of 13786
he of the       13292
Current time : 22:41:34
```

Algorithm:

It is almost same as part 1b, except we find candidate triplet in mapreduce0 and count candidate triplet in mapreduce1. The most different part is how we implement A-priori algorithm to find candidate triplet in part marpper0.

In mapper0, we still need to find candidate pair as part 1a and store them into dictionary(pair_freq_dic) which has “pair” (word1 + “ ”+ word2, the word1 is smaller than word2) as key and “True” (it can be anything, we use true in our case) as value. We loop over the baskets to find triplet where items are all frequent. Then we put them into temporary array and sort them. If the any pair of these 3 items are frequent pair in pair_freq_dic, we store this triplet into dictinary(cand_tri_dic) which has “triplet” as key and “count” as value. Finally, we output those triplet has count over threshold.