CSCI 4707 - Practice of Database Systems
Homework 6
Saksham Goel | goelx029 | 5138568

Q1. Exercise 26.6 Let us develop a new algorithm for the computation of all large itemsets. Assume that we are given a relation D similar to the Purchases table shown in Figure 26.1. We partition the table horizontally into k parts D 1 , ... , D k .

1. Show that, if itemset X is frequent in D, then it is frequent in at least one of the k parts.

Consider some definitions that will be used in the answer
Let a function be NT, that returns the number of relations in a relation R given as parameter = NT(R)
The minsup for the itemsets X is some value = s. (here 0 <= s <= 1)
The number of orders in the relation = N
Let another function be NO, that returns the number of orders in a relation R given as parameter = NO(R)

It means that all the items in the itemset X have appeared together at least (100*s)% times in the relation D. Now considering that we divide the relation D into k parts D1, ….. D k. It means that we can get the entire relation D again by doing a Union operation on all the horizontal partitions. This means that

D = D1 U D2 U D3 …… U Dk

Considering these are horizontal partitions we also know that the number of tuples in each one of them added together is equal to the number of tuples in the entire relation D. Assume - NT() is a function that takes in a relation and returns the number of tuples in that relation then we have a property as follows:

NT(D) = NT(D1) + …… + NT(Dk)
NO(D) = NO(D1) + …… + NO(Dk)

Considering that the partition is based on some kind of key (Order ID, or Customer ID), we know that the all the items that a person pay will be together and not further divided up into smaller chunks. This is important to ensure that we can still have this kind of itemset present in the relations.
Summarizing all the information about the percent times the itemsets have appeared in relation D and how relation has been partitioned we can say that
Percent times all items in X have appeared together in D is >= s%. So in terms of numbers we can say that number of orders with all items in D is greater than s*total number of orders in D. In mathematical terms

NO(D) such that they have all items in X >= NO(D)*s

Now using contradiction assume that the itemset X is not frequent in any of the k parts. So it means that the percent of times all the items in the itemset X have appeared in all the partitioned relations is less than s%. So it means that

For all i, $NO(D_i)$ with all items in itemsets X < $s*NO(D_i)$,
Summing everything up

Sum (For all i from 1 to k, $NO(D_i)$ with all items in itemsets X) < Sum (For all i from 1 to k, $s*NO(D_i)$)

Sum (For all i from 1 to k, NO(D_i) with all items in itemsets X) < s * Sum (For all i from 1 to k, NO(D_i))
because s is a constant

Sum (For all i from 1 to k, NO(D_i) with all items in itemsets X) < s * NO(D)
from the earlier property

NO(D) with all items in itemsets X < s * NO(D)

This statement is contradicting our earlier statement. Hence we can say that there exists at least one partition where itemset X is still frequent.

2. Use this observation to develop an algorithm that computes all frequent itemsets in two scans over D. (Hint: In the first scan, compute the locally frequent itemsets for each part Di , i E {l, ... , k}.)

In the first scan, as stated in the hint we will make a list of all the frequent itemsets for each D_i for all i from 0 to k. Then in the second scan we will just check whether all of these frequent itemsets are frequent in the whole relation D. This means that checking whether the percent times these itemsets have occured in the whole relation more than the minsup percent, because of the direction in which the condition holds. The remaining itemsets are frequent in D.

3. Illustrate your algorithm using the Purchases table shown in Figure 26.1. The first partition consists of the two transactions with transid 111 and 112, the second partition consists of the two transactions with transid 113 and 114. Assume that the minimum support is 70 percent.

First pass through D_1 with transid 111 and 112
The frequent itemsets are :

- {pen}
- {ink}
- {milk}
- {pen, ink}
- {pen, milk}
- {ink, milk}
- {pen, ink, milk}

First pass through D_2 with transid 113 and 114
The frequent itemsets are :

- {pen}

Then finding the itemsets that are not frequent in D by going through all the frequent itemsets found locally

- {pen}
- {ink}
- {milk}
- {pen, ink}
- {pen, milk}

Q2. Assume you are given a document database that contains SIX documents. After stemming, the documents contain the following terms:

Document	**Terms**
1	**car, manufacturer, Honda, auto**
2	**auto, computer, navigation**
3	**Honda, navigation**
4	**Manufacturer, computer, IBM**
5	**IBM, personal, computer**
6	**car, Beetle, VW**

Answer the following questions.
1. Show the result of creating an inverted file on the documents.

The inverted file looks like:

Term (Word)	Number of Documents	Documents
auto	2	<1>, <2>
beetle	1	<6>
car	2	<1>, <6>
computer	3	<2>, <4>, <5>
honda	2	<1>, <3>
IBM	2	<4>, <5>
manufacturer	2	<1>, <4>
navigation	2	<2>, <3>
personal	1	<5>
VW	1	<6>

2. Show the result of creating a signature file with a width of 5 bits. Construct your own hashing function that maps terms to bit positions.

Hashing Function

Term (Word)	Hashed Value
auto	10000
beetle	10000
car	01000
computer	01000
honda	00100
IBM	00100
manufacturer	00010
navigation	00010
personal	00001
VW	00001

Document	Terms	Hash
1	car, manufacturer, Honda, auto	11110
2	auto, computer, navigation	11010
3	Honda, navigation	00110
4	Manufacturer, computer, IBM	01110
5	IBM, personal, computer	01101
6	car, Beetle, VW	11001

3. Evaluate the following boolean queries using the inverted file and the signature file that you created:

- **'car'**
    - Using Inverted File index, car appears in documents - Documents 1 and Documents 6
    - Using Signature File -
      The hash for ‘car’ = 01000

      This appears in Doc 1, 2, 4, 5, 6

      Out of these there are three false matches as follows - Doc 2, 4, 5 because the term ‘computer’ maps to the same hash.

So the results is Doc 1 and Doc 6

- **'IBM' AND 'computer'**
    - Using Inverted File index,
      'IBM' appears in documents - Doc 4 and Doc 5
      'computer' appears in documents - Doc 2, Doc 4 and Doc 5

      So their intersection - Doc 4 and Doc 5
    - Using Signature File -
      The hash for 'IBM' AND 'Computer' = 01100

      This appears in Doc 1, 4, 5

      Out of these there is one false matches as follows - Doc 1 because the term 'honda' and 'car' appear together which maps to the same hash.

      So the results is Doc 4 and Doc 5
- **'IBM" AND 'car'**
    - Using Inverted File index,
      'IBM' appears in documents - Doc 4 and Doc 5
      'car' appears in documents - Doc 1, and Doc 6

      So their intersection - None
    - Using Signature File -
      The hash for 'IBM' AND 'Car' = 01100

      This appears in Doc 1, 4, 5

      All of these three are false matches because the term 'honda' and 'computer' appear together which maps to the same hash.

      So the results is None
- **'IBM' OR 'auto'**
    - Using Inverted File index,
      'IBM' appears in documents - Doc 4 and Doc 5
      'auto' appears in documents - Doc 1, and Doc 2

      So their union - Doc 1, Doc 2, Doc 4 and Doc 5
    - Using Signature File -
      The hash for 'IBM' = 00100

The hash for ‘auto’ = 10000

‘IBM’ according to the hash appears in Doc 1, 3, 4, 5
Of these taking out the false positives (because of Honda) we get Doc 4 and Doc 5.
‘auto’ according to the hash appears in Doc 1, 2, 6
Of these taking out the false positives (because of beetle) we get Doc 1 and Doc 2.

So the results is Doc 1, 2, 4 and 5

- **'IBM' AND 'computer' AND 'manufacturer'.**
    - Using Inverted File index,
      ‘IBM’ appears in documents - Doc 4 and Doc 5
      ‘computer’ appears in documents - Doc 2, Doc 4 and Doc 5
      ‘manufacturer’ appears in documents - Doc 1, and Doc 4

      So their intersection - Doc 4
    - Using Signature File -
      The hash for ‘IBM’ and ‘computer’ and ‘manufacturer’ = 01110

      According to the hash produced the documents might be = Doc 1 and Doc 4.

      Doc 1 is a false positive so the final answer is : Doc 4

4. Assume that the query local against the document database consists of exactly the queries that were stated in the previous question. Also assume that each of these queries is evaluated exactly once.

(a) Design a signature file with a width of 3 bits and design a hashing function that minimizes the overall number of false positives retrieved when evaluating the same queries given in 27.2.3

Hashing Function

Term (Word)	Hashed Value
auto	010
beetle	001
car	010
computer	100
honda	001
IBM	100
manufacturer	100
navigation	001

personal	001
VW	001

Document	Terms	Hash
1	car, manufacturer, Honda, auto	111
2	auto, computer, navigation	111
3	Honda, navigation	001
4	Manufacturer, computer, IBM	100
5	IBM, personal, computer	101
6	car, Beetle, VW	011

I used a python script to compute all possible hash functions and also the false positives from each hashing function for all the queries. It gave me 4 possible hashing functions each giving a total of 14 false positives as the least number of false positives. If you want to see the python script please feel free to go the repository containing that at the link - https://github.com/goelsaksham/umn_csci_4707_query_fp_generator

(b) Design a signature file with a width of 6 bits and a hashing function that minimizes the overall number of false positives.

Hashing Function

Term (Word)	Hashed Value
auto	000100
beetle	000001
car	001000
computer	010000
honda	000001
IBM	100000
manufacturer	000010
navigation	000001
personal	000001
VW	000001

Document	Terms	Hash
1	car, manufacturer, Honda, auto	001111
2	auto, computer, navigation	010101
3	Honda, navigation	000001
4	Manufacturer, computer, IBM	110010
5	IBM, personal, computer	110001
6	car, Beetle, VW	001001

(c) Assume you want to construct a signature file. What is the smallest signature width that allows you to evaluate all queries without retrieving any false positives?
Smallest signature width should be of the order of the number of different words which here is equal to 10 when dealing with all types of query. However considering that if the queries are limited to the type of queries asked before it could be done with width = 6 as done in the previous example without any false positives.

5. Consider the following ranked queries:

- **'car'**
- **'IBM computer'**
- **'IBM car'**
- **'IBM auto'**
- **'IBM computer manufacturer'**

(a) Calculate the IDF for every term in the database.

IDF Values

Term (Word)	IDF
auto	log(6/2) = log(3) = 0.477
beetle	log(6/1) = log(6) = 0.778
car	log(6/2) = log(3) = 0.477
computer	log(6/3) = log(2) = 0.301
honda	log(6/2) = log(3) = 0.477
IBM	log(6/2) = log(3) = 0.477
manufacturer	log(6/2) = log(3) = 0.477

navigation	log(6/2) = log(3) = 0.477
personal	log(6/1) = log(6) = 0.778
VW	log(6/1) = log(6) = 0.778

(b) For each document, show its document vector.

Document Vectors are as follows:

	auto	beetle	car	computer	honda	IBM	manufacturer	navigation	personal	VW
Doc 1	1	0	1	0	1	0	1	0	0	0
Doc 2	1	0	0	1	0	0	0	1	0	0
Doc 3	0	0	0	0	1	0	0	1	0	0
Doc 4	0	0	0	1	0	1	1	0	0	0
Doc 5	0	0	0	1	0	1	0	0	1	0
Doc 6	0	1	1	0	0	0	0	0	0	1

Document Vectors (TF-IDF) are as follows:

	auto	beetle	car	computer	honda	IBM	manufacturer	navigation	personal	VW
Doc 1	0.477	0	0.477	0	0.477	0	0.477	0	0	0
Doc 2	0.477	0	0	0.301	0	0	0	0.477	0	0
Doc 3	0	0	0	0	0.477	0	0	0.477	0	0
Doc 4	0	0	0	0.301	0	0.477	0.477	0	0	0
Doc 5	0	0	0	0.301	0	0.477	0	0	0.778	0
Doc 6	0	0.778	0.477	0	0	0	0	0	0	0.778

(c) For each query, calculate the relevance of each document in the database, with and without the length normalization step.

Query Vectors (TF-IDF) are as follows:

	auto	beetle	car	computer	honda	IBM	manufacturer	navigation	personal	VW
Query 1	0	0	0.477	0	0	0	0	0	0	0

Query 2	0	0	0	0.301	0	0.477	0	0	0	0
Query 3	0	0	0.477	0	0	0.477	0	0	0	0
Query 4	0.477	0	0	0	0	0.477	0	0	0	0
Query 5	0	0	0	0.301	0	0.477	0.477	0	0	0

Document Vectors (TF-IDF) are as follows:

	auto	beetle	car	computer	honda	IBM	manufacturer	navigation	personal	VW
Doc 1	0.477	0	0.477	0	0.477	0	0.477	0	0	0
Doc 2	0.477	0	0	0.301	0	0	0	0.477	0	0
Doc 3	0	0	0	0	0.477	0	0	0.477	0	0
Doc 4	0	0	0	0.301	0	0.477	0.477	0	0	0
Doc 5	0	0	0	0.301	0	0.477	0	0	0.778	0
Doc 6	0	0.778	0.477	0	0	0	0	0	0	0.778

Doc Scores (Used for the length normalization step) (only used for reference, not asked in the question) -

	Length Normalization Score
Doc 1	0.910116
Doc 2	0.545659
Doc 3	0.455058
Doc 4	0.545659
Doc 5	0.923414
Doc 6	1.438097

Without the length normalization step:

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Doc 1	0.227529	0	0.227529	0.227529	0.227529
Doc 2	0	0.090601	0	0.227529	0.090601
Doc 3	0	0	0	0	0

Doc 4	0	0.3183	0.227529	0.227529	0.5456
Doc 5	0	0.3183	0.227529	0.227529	0.3183
Doc 6	0.227529	0	0.227529	0	0

With the length normalization step:

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Doc 1	0.5	0	0.3535	0.3535	0.32287
Doc 2	0	0.21739	0	0.4566	0.166039
Doc 3	0	0	0	0	0
Doc 4	0	0.76376	0.4566	0.4566	1
Doc 5	0	0.5871	0.3510	0.3510	0.4484
Doc 6	0.3977	0	0.2812	0	0

(d) Describe how you would use the inverted index to identify the top two documents that match each query.

We can use the inverted index to find the relevant documents for each query as discussed in the previous part. After getting the relevant documents we can use the relevance score to sort the list of documents and return the top 2.

(e) How would having the inverted lists sorted by relevance instead of document id affect your answer to the previous question?

Having an inverted list sorted by relevance would be helpful in way, such that we would be able to skip the whole sorting business after retrieving the documents as we can use good merging algorithm which would take less complexity time to come up with a list of documents that satisfy the query and also are able sorted in relevance score. This would then be able to return the top 2 relevant documents easily.

(f) Replace each document with a variation that contains 10 copies of the same document. For each query, recompute the relevance of each document, with and without the length normalization step.

Query Vectors (TF-IDF) are as follows:

	auto	beetle	car	computer	honda	IBM	manufacturer	navigation	personal	VW
Query 1	0	0	0.477	0	0	0	0	0	0	0
Query 2	0	0	0	0.301	0	0.477	0	0	0	0

Query 3	0	0	0.477	0	0	0.477	0	0	0	0
Query 4	0.477	0	0	0	0	0.477	0	0	0	0
Query 5	0	0	0	0.301	0	0.477	0.477	0	0	0

Document Vectors (TF-IDF) are as follows:

	auto	beetle	car	computer	honda	IBM	manufacturer	navigation	personal	VW
Doc 1	4.77	0	4.77	0	4.77	0	4.77	0	0	0
Doc 2	4.77	0	0	3.01	0	0	0	4.77	0	0
Doc 3	0	0	0	0	4.77	0	0	4.77	0	0
Doc 4	0	0	0	3.01	0	4.77	4.77	0	0	0
Doc 5	0	0	0	3.01	0	4.77	0	0	7.78	0
Doc 6	0	7.78	4.77	0	0	0	0	0	0	7.78

Doc Scores (Used for the length normalization step) (only used for reference, not asked in the question) -

	Length Normalization Score
Doc 1	91.0116
Doc 2	54.5659
Doc 3	45.5058
Doc 4	54.5659
Doc 5	92.3414
Doc 6	143.8097

Without the length normalization step:

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Doc 1	2.27529	0	2.7529	2.27529	2.27529
Doc 2	0	0.90601	0	2.27529	0.90601
Doc 3	0	0	0	0	0
Doc 4	0	3.183	2.27529	2.27529	5.456

Doc 5	0	3.183	2.27529	2.27529	3.183
Doc 6	2.27529	0	2.27529	0	0

With the length normalization step:

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Doc 1	0.5	0	0.3535	0.3535	0.32287
Doc 2	0	0.21739	0	0.4566	0.166039
Doc 3	0	0	0	0	0
Doc 4	0	0.76376	0.4566	0.4566	1
Doc 5	0	0.5871	0.3510	0.3510	0.4484
Doc 6	0.3977	0	0.2812	0	0

Q3. You are in charge of the Genghis ('We execute fast') search engine. You are designing your server cluster to handle 500 Million hits a day and 10 billion pages of indexed data. Each machine costs $1000, and can store 10 million pages and respond to 200 queries per second (against these pages).

1. If you were given a budget of $500,000 dollars for purchasing machines, and were required to index all 10 billion pages, could you do it?

$500,000 is equivalent to 500000/1000 = 500 machines. This is equivalent to 500 * 10 Million pages = $5 * 10^9$ pages. Which is equivalent to 5 billion pages. However the requirement is for 10 billion pages and thus we cannot do it.

2. What is the minimum budget to index all pages? If you assume that each query can be answered by looking at data in just one (10 million page) partition, and that queries are uniformly distributed across partitions, what peak load (in number of queries per second) can such a cluster handle?

Minimum cost would be equivalent to $1,000,000 = 1 Million Dollars. This would allow us to have enough machines to index around 10 billion pages in 10 million page partitions on 1000 machines. The server would be able to handle around 200 * 1000 = 200,000 queries per second.

3. How would your answer to the previous question change if each query, on average, accessed two partitions?

Then the answer would be around 100,000 queries per second. Because two machines together are expected to respond to 200 queried per second.

4. What is the running budget required to handle the desired load of 500 million hits per day if all queries are on a single partition? Assume that queries are uniformly distributed with respect to time of day.

Assumption - That the facility already has stored the 10 billion pages.
500 million hits per day = 5788 queries per second. Considering they are using one partition each, we need 5788/200 machines = 29 machines. So the budget = 29*1000 = $29,000.
If we do want to account for the 10 billion pages, the budget should be $1 million, which will give us more machines that needed to compute the queries while making sure that the machines are able to save all the 10 billion pages of indexed data.

5. Would your answer to the previous question change if the number of queries per day went up to 5 billion hits per day? How would it change if the number of pages went up to 100 billion'?

Assumption : That we have already been able to index all the pages (10 Billion)

a) Yes, the answer will change as follows:
5 billion hits per day = 57880 queries per second. Considering they are using one partition each, we need 57880/200 machines = 290 machines. So the budget = 290*1000 = $290,000.

b) The answer should not change, because the running budget does not depend on the number of

pages, because we are assuming that we have enough machines already that can index all the pages.

6. Assume that each query accesses just one partition, that queries are uniformly distributed across partitions, but that at any given time, the peak load on a partition is upto 10 times the average load. What is the minimum budget for purchasing machines in this scenario?

Assumption : That we have already been able to index all the pages (10 Billion)

Assuming that the number of hits per day = 500 million (Mentioned in the question at the starting)

500 million hits per day = 5788 queries per second. Considering they are using one partition each, we need 5788/200 machines = 29 machines. So the budget = 29*1000 = $29,000. However, because the peak load could still be 10 times that on the partition, we need around 10 times the machines, so the answer becomes 29*10*1000 = $290,000

7. Take the cost for machines [take the previous question and multiply it by 10 to reflect the costs of maintenance, administration, network bandwidth, etc. This amount is your annual cost of operation. Assume that you charge advertisers 2 cents per page. What fraction of your inventory (i.e., the total number of pages that you serve over the course of a year) do you have to sell in order to make a profit?

The cost of machines after accounting for the other costs → $2,900,000
So number of pages required = $2.9*10^6/0.02 = 1.45*10^8$.
We have around 10^{10} pages. So the fraction = $1.45 * 10^8/10^{10} = 1.45/100 = 1.45\%$