CSCD43 report

Name: Zhuo Gu

Utorid: guzhuo

Student #: 1000047068

Blue line stands for 2 hash functions

Red line stands for 3 hash functions

Yellow line stands for 4 hash functions

According to the reference that we have,
the false positive rate p = (1 - e^(kn/m))^k

$$\left(1 - e^{-\frac{kn}{m}}\right)^k$$

m is number of bits in the array
k is number of hash functions
n is the number of inserted elements
based on the above formula, then we have two following formula:
k = (m/n)(ln(2)

$$k = \frac{m}{n} \ln 2,$$

m = (-n(lnp)/(ln(2)^2))

$$m = -\frac{n \ln p}{(\ln 2)^2}.$$

if n is increasing, then
with the increase of m, k is generally increasing too, but all of them are depends on p.

In my program, I calculate the **total positive, and true positive and true negative.**
false positive = total positive - true positive
tuples dropped = true negative + false positive
false positive rate p = false positive / tuples dropped

1) Based on the first question

Fix outer relation R = 7000

Fix Bloom Filter size = 47925

SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 3000 AND S.ID < 4000

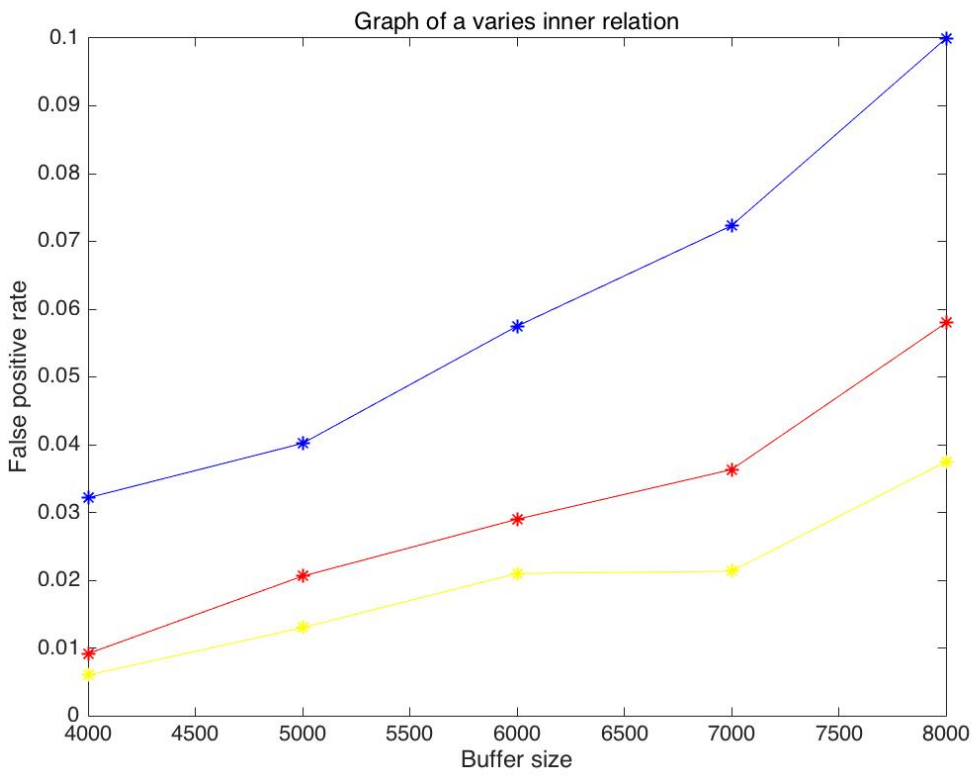
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 3000 AND S.ID < 5000

SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 3000 AND S.ID < 6000

SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 3000 AND S.ID < 7000

SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 3000 AND S.ID < 8000

	4000	**5000**	**6000**	**7000**	**8000**
2	0.0322	0.0402	0.0575	0.0723	0.0999
3	0.0092	0.0206	0.0290	0.0363	0.0580
4	0.0060	0.0156	0.0210	0.0213	0.0375



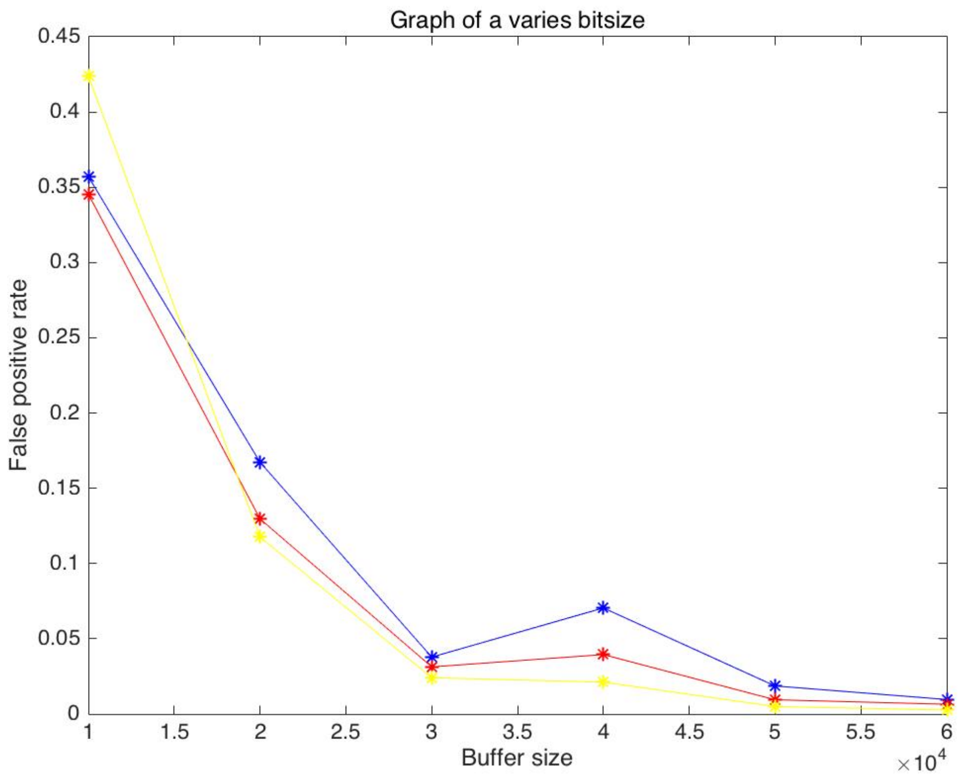
Graph 1

As we can see from the graph, with the increasing number of inner inputs, the false positive rate is increasing, because it decreases the total number of tuples that dropped from second condition S.ID > # such as first query need to drop 10000 - 4000 = 6000 tuples, and with more hash functions, it flipped more bit in bit array, which increase the possibility of dropping the right tuples, so the false positive is decreasing as well, but there is only a small amount of decreasing in false positive, so

the overall formula false positive rate = false positive/total dropped tuples is increasing as the graph shows.
In conclusion, the interesting result is that if we increase the number of inner inputs, then the false positives would increase, because it need to drop less tuples than before.

2) Based on the second question
Fixed inner S: 4000
Fixed outer R: 7000
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 3000 AND S.ID < 4000

	10000	**20000**	**30000**	**40000**	**50000**	**60000**
2	0.3571	0.1673	0.0378	0.0705	0.0187	0.0097
3	0.3454	0.1296	0.0313	0.0395	0.0095	0.0065
4	0.4239	0.1176	0.0242	0.0212	0.0050	0.0027



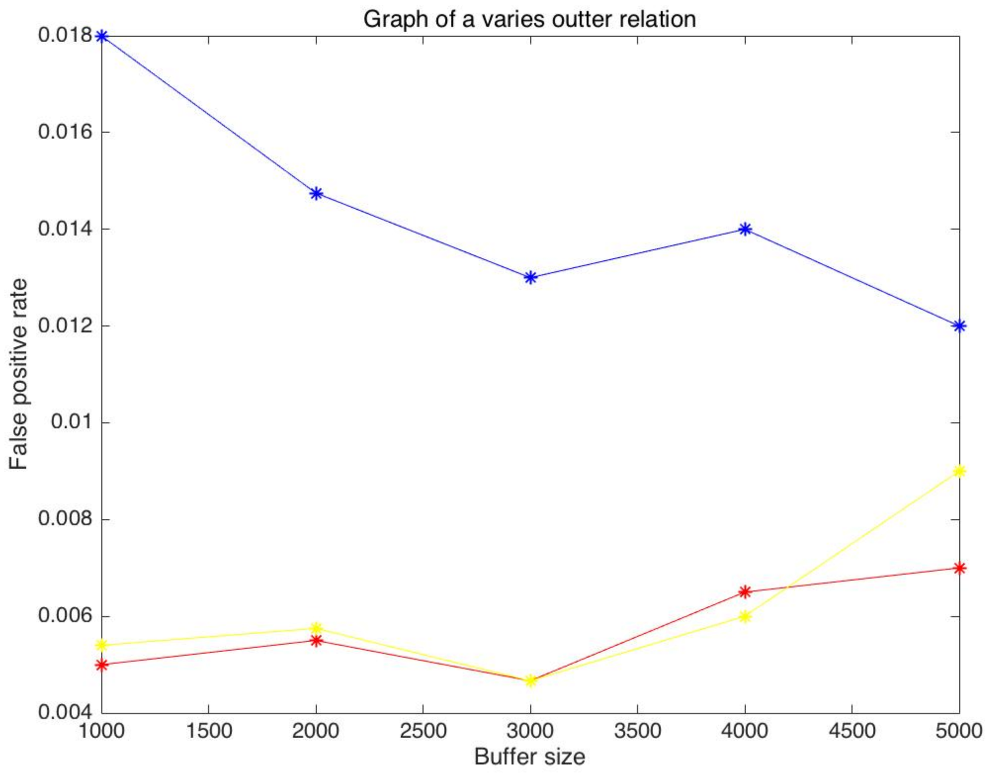
Graph 2
As we can see from the graph, with the increasing size of bit array, the rate of false positives is decreasing dramatically, because if we increase the size of bit array, it means that there will be more positions for Bloom filter to flip the bit. Moreover, if we use more hash functions to flip the bit, it will decrease the probability of not dropping the wrong one, which is the false positive. So the graph basically shows the correct performance of increasing bit array size.

In conclusion, the interesting result is that if we increase the size of bit array dramatically, it will increase the performance of Bloom filter dramatically.

3) Based on the third question
Fixed Bloom filter size: 47925
Fixed outer relation S: 4000

SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 1000 AND S.ID > 6000
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 2000 AND S.ID > 6000
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 3000 AND S.ID > 6000
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 4000 AND S.ID > 6000
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 5000 AND S.ID > 6000

	9000	**8000**	**7000**	**6000**	**5000**
2	0.0180	0.0148	0.0127	0.0140	0.0120
3	0.0050	0.0055	0.0047	0.0065	0.0070
4	0.0054	0.0058	0.0047	0.0060	0.0090



Graph 3

as the graph shows, with the decreasing of outer inputs, the rate of false positive rate is a little wired, but the result is that if we use more hash functions, then we can reduce the rate dramatically. The reason is that the first condition R.ID > 1000 will consider the bigger 9000 tuples, but the second condition S.ID > 6000 means that it still need to drop 6000 - 1000 = 5000 tuples for returning the right result. So with the decreasing of outer inputs, there will be less tuples for dropping, e.g. 6000 - 2000 = 4000, 6000 - 3000 = 3000, etc.…then the total number of dropped tuple is decreasing, and the number of false positive is slightly decreasing. So the overall rate is not that clear to see, because both of total numbers of dropped tuple and false positive are decreasing, so the rate is fluctuating.

1) How many hash functions would you use?

   According to above graphs, the number of hash functions is depending on the size of bit array, so according to all graphs, if we have more hash functions, then we have the lowest rate of false positive. But if we have too much hash functions, it might cause some unstable rates of false positive, such as the rate in Graph 3, if we use 4 hash functions, the rate is increasing faster than the rate of using 3 hash functions, and it is relatively steady to use 3 hash functions according to Graph 3. So I would like to choose 3 hash functions.

2) What would be the size of your bit array?

   According to above graphs, the size of bit array m is depending on the rate of false positive p and elements n, it we choose a smaller p, and static n, then m will need to increase dramatically, but according to Graph 2, the rate is relatively steady around 50000 bits, so I choose 47925 bits as my size of bit array, it is calculated by set p = 0.01 with the average of inserted elements n = 5000.

Strategies that I used to implement:

Based on the formula that I mentioned above, the size of bit array m and number of hash functions k is depending on the rate of false positive. Thus I decided to set false positive rate p to be 0.01, because it is relative steady from the result I analyzed above. However, if the rate of false positive is smaller, it will create a larger size of bit array, but I only have 5 hash functions in my implementation, so it might cause unstable performance for Bloom filter, so finally, according to the graphs, I use 0.01 as the rate of false positive. Moreover, we only estimate the number of tuples in data, not the actual number of tuples, so it also might affect the performance of bloom filter.

In conclusion, the overall strategy is set the p to be 0.01, and calculate m and k based on the number of inserted elements n, then use the filter to drop tuples fast.