CSCD43 HW2 Report

I. Group Information

- Student 1: Zhitao Xu, Student number: 1006668697
- Student 2: Lianting Wang, Student number: 1007452374

II. Preparation

1. Database table structure

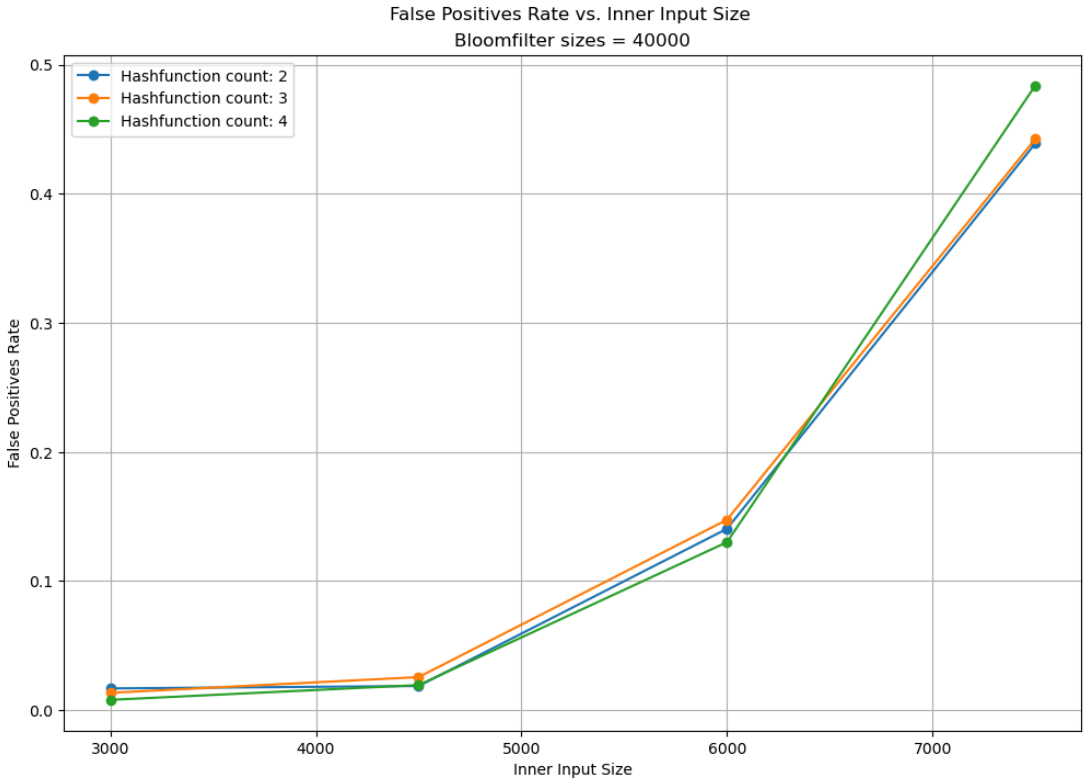
- CREATE TABLE R (ID INTEGER PRIMARY KEY, A INTEGER, B INTEGER, C INTEGER);
- CREATE TABLE S (ID INTEGER PRIMARY KEY, A INTEGER, B INTEGER, C INTEGER);

2. The formula used to compare the diagram:

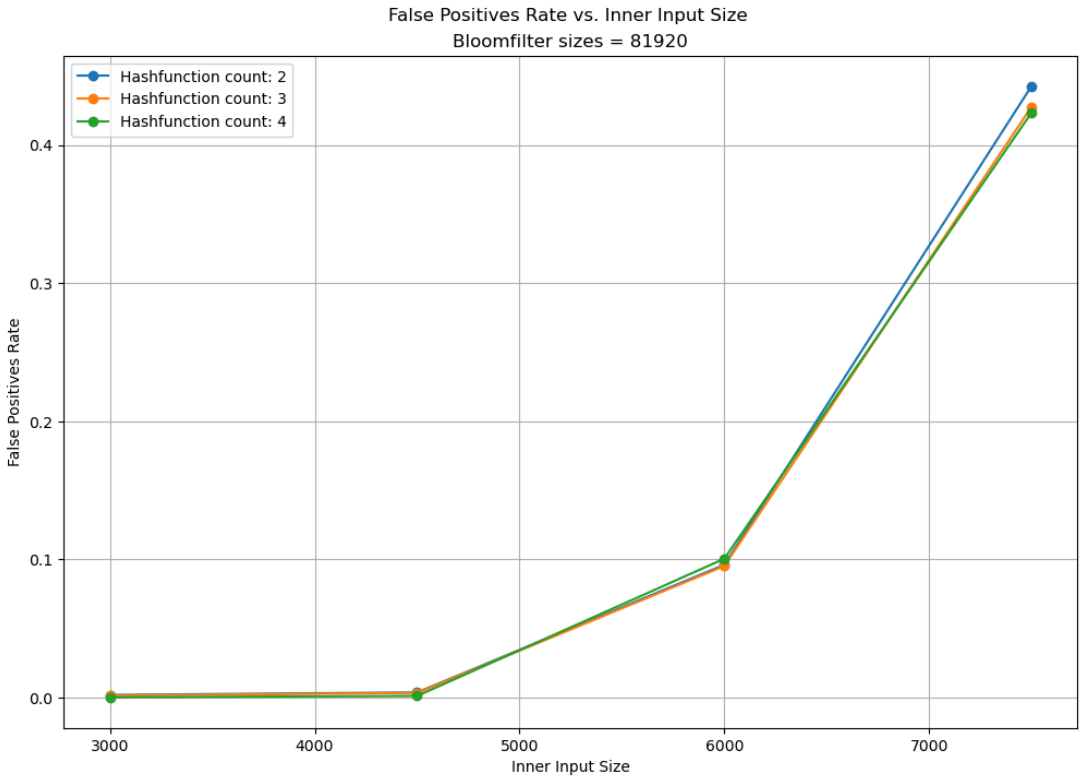
- False Positives (FP): a tuple that does not join with anything is let to pass
- True Positives (TP): a tuple that could join with another tuple is let to pass
- False Negatives (FN): a tuple that could join with another tuple is dropped
- True Negatives (TN): a tuple that could not join with anything and dropped
- False Positive Rate (FPR): FP / (FP + TN)

III. Diagram and Compare

- Diagram 1: Keeping the size of the bloom filter fixed, vary the size of the inner input of the hash-join, by changing the selectivity condition on the inner, and plot the false-positive rates when 2, 3 and 4 hash functions are used.
- We choose to have the bloom filter fixed at 40000 bits and 80000 bits, fix the outer tuple size, and vary the size of the inner join tuples to plot the following diagram:
- Diagram:

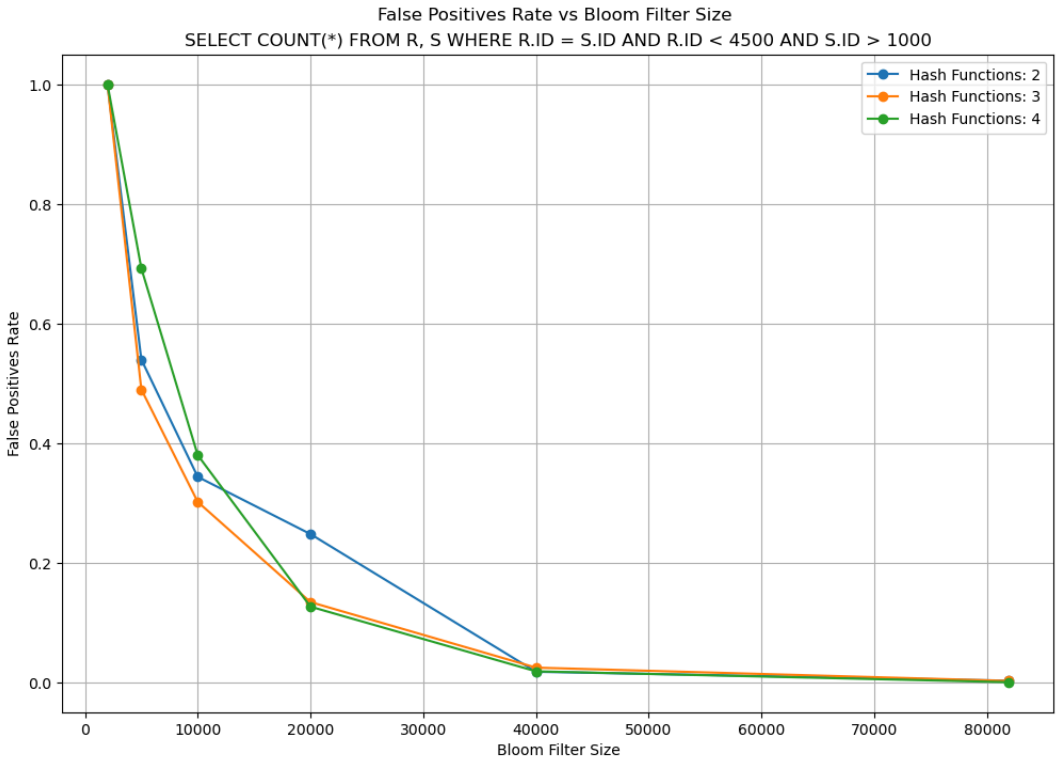


-

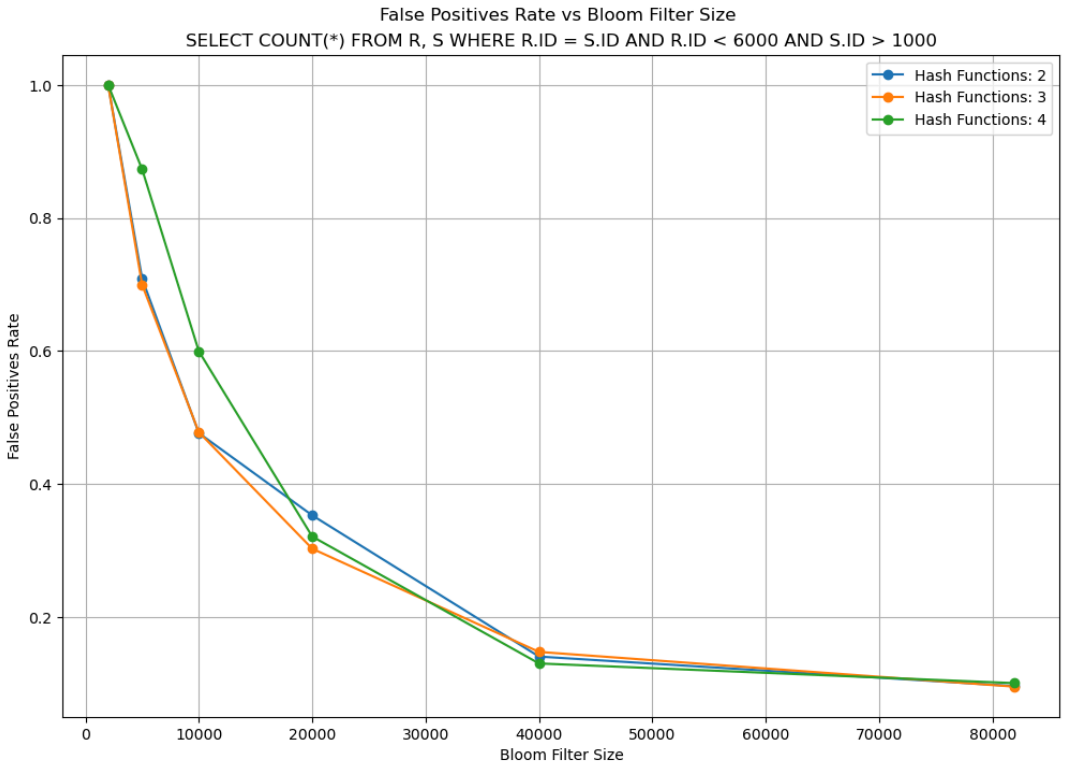


-
- Explain: The x-axis is the size of the inner join tuples, and the y-axis is the false positive rate stated in part 2.
- Workload we use:
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 3000 AND S.ID > 1000;
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 4500 AND S.ID > 1000;
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 6000 AND S.ID > 1000;
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 7500 AND S.ID > 1000;
- Analyze:
    - When the inner input size is modest, the false positive rates are relatively low for all hash function counts, which is expected because the Bloom filter has enough capacity to accurately represent the data. As the inner input size increases, we see the false positive rate rise for all hash function counts due to the higher likelihood of bit collisions within the filter.
    - When the inner join tuple size is above 6000, the false positive rate becomes significant and when we further increase it to 7500, the false positive becomes high. This indicates that the bloom filter might be not that efficient as the inner size tuples increase.
    - Moreover, three hash functions tend to offer a more favourable performance, often yielding the lowest false positive rate across the tested range. (Both 2 and 4 seem to have some cases that lead to a high false positive rate). This implies that using three hash functions is a more balanced approach, effectively reducing false positives.

- Diagram 2: Keeping the size of the inner input fixed, vary the size of the Bloom filter and plot the false-positive rates when 2, 3 and 4 hash functions are used.
- We choose to have the inner join tuple size fixed to be 4500 tuples and 6000 tuples, fix the outer tuple size and vary the size of the bloom filter to plot the following diagram:
- Diagram:



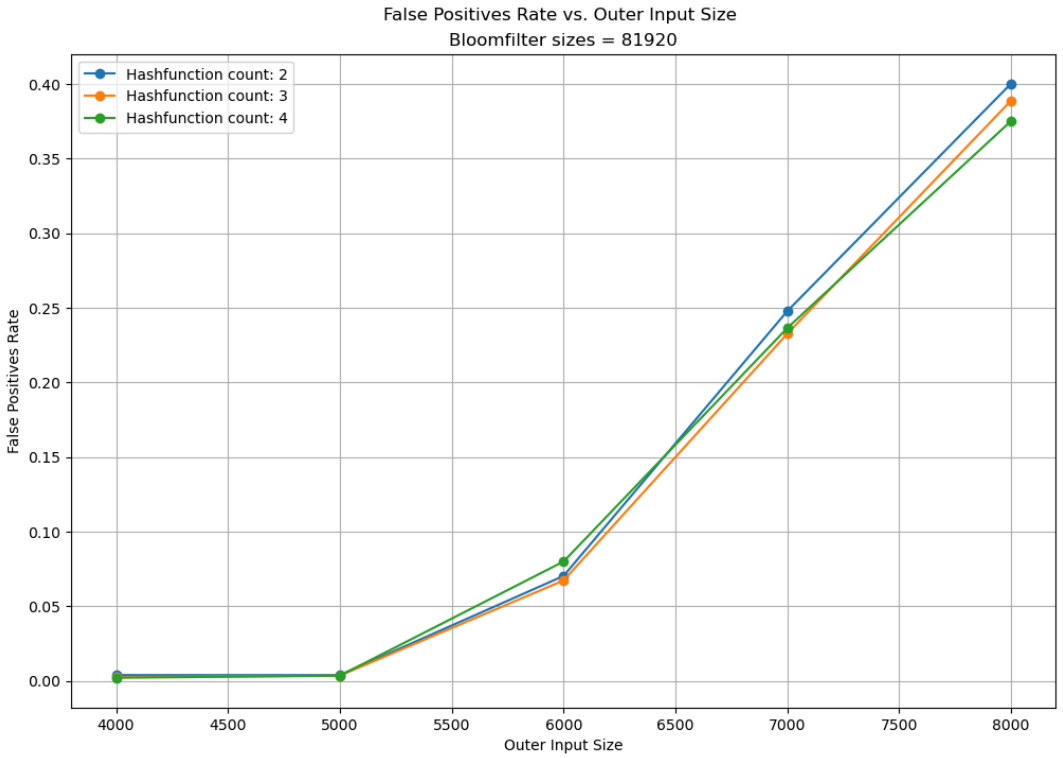
-



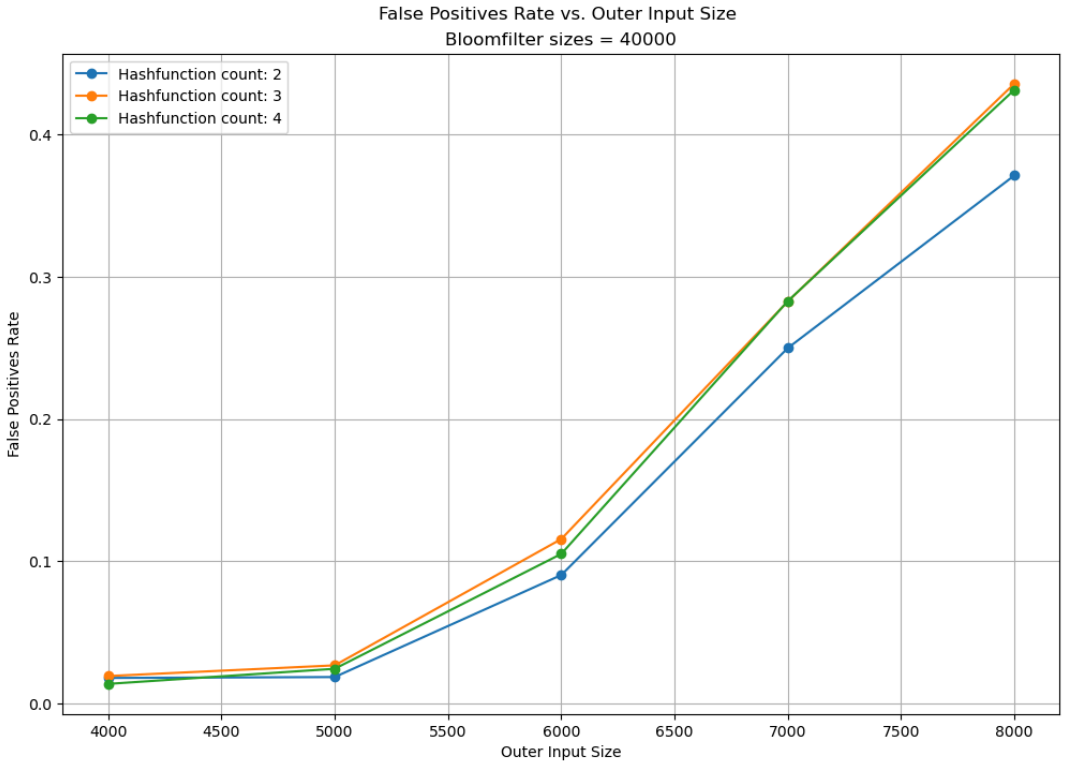
-

- Explain: The x-axis is the size of the bloom filter in bits, and the y-axis is again the false positive rate
- Workload we use:
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 4500 AND S.ID > 1000;
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 6000 AND S.ID > 1000;
- Analyze:
    - When the Bloom filter size is quite small, the false positive rate is high regardless of the number of hash functions used. This makes sense because a small bit array would quickly fill up, causing the Bloom filter to become ineffective. As the size of the Bloom filter grows, the false positive rates for all numbers of hash functions start to drop. From the graphs, it's clear that at a filter size of about 20,000 bits, using only two hash functions results in a worse false positive rate compared to using three or four hash functions. This indicates that two hash functions are not as efficient at this size.
    - Additionally, three hash functions seem to consistently offer the best performance, yielding the lowest false positive rate across almost all sizes of Bloom filters. This suggests that three hash functions might be the sweet spot, providing a good balance for the conditions tested, as it reduces false positives effectively without the extra complexity of more hash functions.
    - Lastly, once the Bloom filter size reaches around 40,000 bits, adding more hash functions doesn't significantly improve the false positive rate. The performance of two, three, and four hash functions becomes similar as the lines on the graph converge, indicating that increasing the filter size further has diminishing returns on improving accuracy.

- Diagram 3: Keeping the size of the inner, and the size of the Bloom filter fixed, vary the outer size (again, by changing selection conditions), and plot the false-positive rates when 2, 3 and 4 hash functions are used.
- We choose to have the inner join tuple size fixed to be 3000 tuples, the bloom filter size to be 40000 bits and vary the size of the outer join tuples to plot the following diagram:
- Diagram



-



-

- Explain: The x-axis is the size of the outer join tuples, and the y-axis is the false positive rate
- Workload we use:
  - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 7000 AND S.ID > 2000; // (the case outer input size 8000)
  - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 7000 AND S.ID > 3000; // (the case outer input size 7000)

    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 7000 AND S.ID > 4000; // (the case outer input size 6000)
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 7000 AND S.ID > 5000; // (the case outer input size 5000)
    - SELECT COUNT(*) FROM R, S WHERE R.ID = S.ID AND R.ID < 7000 AND S.ID > 6000; // (the case outer input size 4000)
- Analyze:
    - When the outer input size is modest, the false positive rates are relatively low for all hash function counts, which is expected because the Bloom filter has enough capacity to accurately represent the data. As the outer input size increases, we see the false positive rate rise for all hash function counts due to the higher likelihood of bit collisions within the filter.
    - When the outer join tuple size is above 6000, the false positive rate becomes significant and when we further increase it to 7500, the false positive becomes high. This indicates that the bloom filter might be not that efficient as the outer size tuples increase.
    - The number of hash functions seems to be difficult to determine. During testing, we found that with a Bloom filter size of 40000, two hash functions provide a lower false positive rate. The reason for this is that a bloom filter size of 40000 might be too small for the outer input size of 8000, and an increase in the number of hash functions will increase the collision rate at the size.

IV. Choose A Plan

- Plan: If we need to choose a static plan, the number of the hash function will be 3 and the size of the bit array will be 10kb (81920 bits).
- Since 3 hash functions with 81920 bits perform the best on all of the previous diagrams/tables, and each of them is not too big or complicated to implement or maintain.

V. Dynamic bloom filter size implementation

- We reviewed a source from the Computer Science Stack Exchange (https://cs.stackexchange.com/questions/149136/optimal-parameters-for-a-bloom-filter) that discusses optimal parameters for a Bloom filter. There, we learned about the general formula to calculate the best dynamic Bloom filter size and the optimal number of hash functions. The formula for the optimal bit array size is (−nlogϵ/log(2)^2), and for the optimal number of hash functions, it is ((m/n) * ln(2)), where ϵ is the expected false positive rate set at 5% for our purposes.
- Based on this information, we implemented functions named calculateBloomFilterSize and calculateBloomFilterHashFunctionSize to perform these calculations during the initialization of the bit array. We also set constraints with MAX_BLOOMFILTER_SIZE = 81920 and MAX_HASHFUNCTIONS = 4 due to potential limitations in filter size and the number of hash functions available. Also, we analyzed that the estimated row might be off by around 10%, so we decided not to shift our estimated rows because it wouldn't change the result's false positive rate much.

- In practice, we observed that the Bloom filter size adjusted accordingly with the size of the outer join tuples, decreasing for smaller sizes and increasing for larger ones, aligning with our expectations. (One thing we also observe is that the maximum size is seldom achieved, which means the dynamic bloom filter size can reduce the memory pressure compared to static 81920 bits.) Regarding the number of hash functions, despite previous tests indicating that 3 hash functions provided the most balanced performance, our optimal function always recommends 4. This may be a result of our hash functions not being perfectly uniform. Nonetheless, four hash functions also performed well, so we decided to continue using the formula to calculate the optimal number of hash functions.

VI. Hash Function we implemented

- In total, we implemented five hash functions and have them in a list (hashFunctions[] = {HashFunctionFNV, HashFunctionPJW, HashFunctionSDBM, HashFunctionAP, HashFunctionDEK}) to store them.
- The main source of the hash function is https://www.partow.net/programming/hashfunctions/ and Wikipedia, the pseudo-code is before each hash function itself.