Due Date: April 26, 2024. Time: 11:00PM

Problem 1: [20 points]
Implement an algorithm to evaluate $q(A, B, C) : -R_1(A, B), R_2(B, C)$. Assume that each attribute has domain $\mathbb{Z}$, i.e., the set of integer numbers. There are many ways to implement this but one of the most common way is to use hashing. For example, you can hash the tuples of R_2 having as keys the numbers in attribute B and as values a list of tuples in R_2 that satisfy the key. For example, if h is a hush function then for $b \in \mathbb{Z}$, $h[b]$ returns $\{t \in R_2 \mid \pi_B(t) = b\}$. Then, we go through each tuple t' in R_1. We run a query on the hashmap using $\pi_B(t')$ and report all tuples in R_2 that can be joined with R_1. We report all the results in the join.

Create a dataset having 10 tuples in R_1 and 10 tuples R_2. Run your algorithm and explain why this is correct.

In the report, you should explain how you implement your algorithm and why your implementation is correct.

Problem 2: [40 points]
The goal is to implement an algorithm (for example using a simplified version of Yannakakis algorithm) to evaluate line join queries in $O(N + OUT)$ time. For example, $q(A_1, A_2, A_3, A_4) : -R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$, is a 4-line join. For simplicity assume that i) the domain of each attribute is $\mathbb{Z}$, ii) each relation has N tuples, iii) each relation has exactly two attributes and they are joined as shown in the example above, and iv) you should focus on k-line join queries for $k \leq 10$.

In the report you need to explain how you implement your algorithm.

Problem 3: [20 points]
Implement a simpler algorithm to evaluate a line join query as follows. Assume $q(A_1, \ldots, A_{k+1}) : -R_1(A_1, A_2), R_2(A_2, A_3), \ldots, R_k(A_k, A_{k+1})$. We first compute $R_{1-2} = R_1 \bowtie R_2$ using the implementation from Problem 1. Then you compute $R_{1-3} = R_{1-2} \bowtie R_3$ and so on. In the end, you compute $R_{1-k} = R_{1-(k-1)} \bowtie R_k$.

Problem 4: [20 points]
Create a random dataset for the 3-line join as follows. In each relation, we have 100 tuples. In relation R_1 we add 100 tuples (i, x) where $i = 1, \ldots, 100$, and x is a random integer between 1 and 5000. In relation R_2 we add 100 tuples (y, j) where $j = 1, \ldots, 100$, and y is a random integer between 1 and 5000. Finally, in R_3 we add the tuples (ℓ, ℓ), where $\ell = 1, \ldots, 100$.

Run your implementation from Problem 2 in this dataset.

Run you implementation from Problem 3 in this dataset.

Show and explain the results in the report. Do they return the same results? What is the running time?

Problem 5: [20 points]
Create a dataset for 3-line query as follows. We add 1000 tuples in R_1 with values $(i, 5)$, where $i = 1, 2, \ldots, 1000$. Then we add 1000 more tuples in R_1 with values $(i, 7)$, where $i = 1001, 2, \ldots, 2000$. Finally, we add the tuple $(2001, 2002)$ in R_1. Take a random permutation on the tuples in R_1 (tuples have arbitrary position in the table).

Then we construct R_2 as follows. We add 1000 tuples in R_2 with values $(5, i)$, where $i = 1, 2, \ldots, 1000$. Then we add 1000 more tuples in R_2 with values $(7, i)$, where $i = 1001, 1002, \ldots, 2000$. Finally, we add the tuple $(2002, 8)$ in R_2. Take a random permutation on the tuples in R_2 (tuples have arbitrary position in the table).

Finally, in R_3 add 2000 random tuples (x, y) such that x is a random number between 2002 and 3000. The value y is any random number between 1 and 3000. In the end, add the tuple $(8, 30)$ in R_3. Take a random permutation on the tuples in R_3 (tuples have arbitrary position in the table).

Run your implementation from Problem 2 in this dataset.

Run you implementation from Problem 3 in this dataset.

Show and explain the results in the report. Do they return the same results? How fast are the two implementations and why?

Problem 6: [More BONUS][20 points]
Run the 3-line join using the dataset of Problem 5 on MySQL. Are the results the same with the other implementations? What is the running time? Is the actual running time closer to the running time of the implementation from Problem 2 or from Problem 3 and why?