Due Date: December 10. Time: 11:00PM

1. (10 points) **Problem 1.** 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 integers. There are many ways to implement this but one of the most common is to use hashing. For example, hash the tuples of R_2 using keys from attribute B and values as the list of tuples in R_2 that share the key. If h is a hash map, then for $b \in \mathbb{Z}$,

$$h[b] \ = \ \{\, t \in R_2 \mid \pi_B(t) = b \,\}.$$

Then iterate over each tuple $t' \in R_1$, probe the hash map with $\pi_B(t')$, and report all tuples in R_2 that can be joined with t'. Report all results in the join.

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

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

2. (20 points) **Problem 2.** The goal is to implement an algorithm (e.g., a simplified version of the Yannakakis algorithm) to evaluate *line join* queries in $O(N + \mathrm{OUT})$ 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 joins as shown above; and (iv) you focus on k-line joins for $k \leq 10$.

In the report, explain how you implement your algorithm.

3. (10 points) **Problem 3.** Implement a simpler algorithm to evaluate a line join query as follows. Assume

$$q(A_1, \ldots, A_{k+1}) \ :- R_1(A_1, A_2),\, R_2(A_2, A_3),\, \ldots,\, R_k(A_k, A_{k+1}).$$

First compute $R_{1\text{-}2} = R_1 \bowtie R_2$ using the implementation from Problem 1. Then compute $R_{1\text{-}3} = R_{1\text{-}2} \bowtie R_3$, and so on, until $R_{1\text{-}k} = R_{1\text{-}(k-1)} \bowtie R_k$.

4. (10 points) **Problem 4.** Create a random dataset for the 3-line join as follows. In each relation, include 100 tuples.

   - In R_1, add 100 tuples (i, x) where $i = 1, \ldots, 100$ and x is a random integer in $[1, 5000]$.
   - In R_2, add 100 tuples (y, j) where $j = 1, \ldots, 100$ and y is a random integer in $[1, 5000]$.
   - In R_3, add the tuples (ℓ, ℓ) for $\ell = 1, \ldots, 100$.

Run your implementation from Problem 2 on this dataset.

Run your implementation from Problem 3 on this dataset.

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

5. (10 points) **Problem 5.** Create a dataset for a 3-line query as follows.

   - In R_1:
     - Add 1000 tuples $(i, 5)$ for $i = 1, 2, \ldots, 1000$.
     - Add 1000 tuples $(i, 7)$ for $i = 1001, 1002, \ldots, 2000$.
     - Add the tuple $(2001, 2002)$.
     - Apply a random permutation to the tuples in R_1.
   - In R_2:
     - Add 1000 tuples $(5, i)$ for $i = 1, 2, \ldots, 1000$.
     - Add 1000 tuples $(7, i)$ for $i = 1001, 1002, \ldots, 2000$.
     - Add the tuple $(2002, 8)$.
     - Apply a random permutation to the tuples in R_2.
   - In R_3:
     - Add 2000 random tuples (x, y) with $x \in [2002, 3000]$ and $y \in [1, 3000]$.
     - Add the tuple $(8, 30)$.
     - Apply a random permutation to the tuples in R_3.

   Run your implementation from Problem 2 on this dataset.

   Run your implementation from Problem 3 on this dataset.

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

6. (10 points) **Problem 6.** Run the 3-line join using the dataset of Problem 5 on MySQL. Are the results the same as 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?

7. (30 points) **Problem 7.** Consider the query $q(A_1, A_2, A_3, A_4, A_5, A_6) : -R_1(A_1, A_2), R_2(A_2, A_3)$ $R_3(A_1, A_3) R_4(A_3, A_4) R_5(A_4, A_5) R_6(A_5, A_6) R_7(A_4, A_6)$.

   Implement the GenericJoin algorithm specifically for query q. In the report, show your algorithm both in pseudocode and with a written explanation describing its logic and steps. What is the asymptotic running time?

   Implement the Generalized Hypertree Width algorithm for query q. In the report, show your algorithm both in pseudocode and with a written explanation describing its logic and steps. What is the asymptotic running time?

   Implement the Fractional Hypertree Width algorithm for query q. In the report, show your algorithm both in pseudocode and with a written explanation describing its logic and steps. What is the asymptotic running time?

   Download the dataset QueryRelations

   Run all your three algorithms on this dataset. In the report, show the running time of each of the implemented algorithms. Compare their empirical running times with the asymptotic running times.