Gamma Joins

Kaivan Wadia, Mark Gray

Pre-Set Up:

We began by both separately downloading the Parallel sort algorithm and understanding how it was implemented. This gave us a basis for understanding how to do the Gamma Joins set up. Understanding the way that ThreadList was set up was key to figuring out how the classes would interact together.

Set Up:

We started building the project by downloading the basicConnector and gammaSupport packages from the assignment page. Separated them out into their own Java packages and understood how they were supposed to work together. First we implemented ReadRelation and Print, as recommended by the instructions to ensure that we were reading and printing the tuples correctly. We also wrote the rest of the helper classes i.e Sink, SinkMap, PrintMap and DoNothing. These were very tedious but essential and helpful for testing to write.

Gamma Join Classes:

After we had a starting point and the initial Print class test done we started working on implementing all the classes described in the first hint. We started of by creating the building blocks of Gamma i.e. Bloom , BloomFilter and Hjoin. After writing each class we wrote extensive test cases for each of those classes. Each class extended the Thread class and was added to the ThreadList inside of its constructor to make it callable from the test cases. Next we worked on the splitting classes (HSplit and SplitMap) and the merging classes (Merge and MergeMap). These were a little trickier to write and especially test. Our testing strategy involved writing simple regression tests that determined that the output generated by each class matched the correct out put given by that function.

Gamma Join Implementation:

Finally after verifying the gamma join base classes we implemented the ArrayConnectors sub-classes. Each of these used several of the base classes and were described in detail in the powerpoint. Building the classes was not difficult considering the superb presentation in class. The Gamma class was a bit of a beast though. It used many of the gamma join classes but ended up cumulatively more optimized than the HjoinBloomFilters class. After we got the entire package gammaIntegration working we developed several JUnit regression tests for each class to make sure that we are generating correct output.

Gamma Join Notes:

We found that implementing Gamma Join, and most software, in this way adds a layer of file complexity to it, but also has many benefits as well. The file structure becomes very complex for implementing just the Gamma Join because of the way we split it into many parts. However, by splitting this up in this way we also made the code much more understandable and with the dataflow diagram putting the pieces together was relatively simple. By doing Gamma Join in this way we could also test each piece of the algorithm for correctness and in turn be completely sure that the entirety of our algorithm was working correct. The design by transformation approach to implementing the Inner Join of two tables was relatively simple and made it easy to implement the entire design.