Titouan CAZIN
Guillaume REQUENA

Groupe AJ

20/01/20

DBSYS
IEJOIN Lab Report

README

In this report, we will help you understand how we manage to complete the different tasks.
First let's have a quick overview about how far we went in this assignement:

We successfully implemented: Task 1a – Task1b – Task 2a – Task 2b – Task 2c
We have bugs and we didn't manage to solve them: Task 2c1 – Task 2c2
We didn't manage to arrive until this task: Task 2d

How to run the code?

You have to change the Query methods to be used in the runTests methods and you can only run the query one by one by modifying the runTests().
Then go in the file directory src and type "make test" in the Shell and you will have the result for the query in the Output/output_file repertory.

What can you modify in the txt file queries?

In every txt file, you can modify the numbers only, not the letters associated to the table. We didn't implement it yet. For the query 1b you can also modify the AND with an OR.
However, you can't modify the AND on the query 2b.

Global things about our code

For task 2a and 2b we used the sort tuple function and then we convert it into ArrayList to compute the IESelfJoin algorithm correctly.
We put all the queries class and condExpr class in a file called JoinTest.java.
For each of the query 2a, 2b and 2c, we created a IESelfJoinnameofthequery.java file.

Task 1a

Assumptions
All tables should be in the format of 4 columns of integer.
When modifying the text file of the query1a, you can’t modify the letters, only the numbers.

For the first task, we implemented the Nested Loops Join algorithm with single predicate inequality joins. We followed the structure of the original JoinTest.java that was used for the boats and sailors queries. We first created the table R and S then we loaded them with the R.txt and S.txt. Then we created the Query1a_CondExpr() and the Query1a() methods following the example of the former Sailor() method. In our method we call the nlj algorithms computed in the NestedLoopJoins.java file.

Results
You can see the results of the query in the Output directory in the file output_query1a.txt.

Task 1b

Assumptions
All tables should be in the format of 4 columns of integer.
When modifying the text file of the query1a, you can’t modify the letters, only the numbers, but you can modify the AND with a OR.

For this second task, we implemented the Nested Loops Join algorithm with two predicates inequality joins. We followed the structure of methods we created for query1a but we just added an other CondExpr in the Query1a_CondExpr().

Results
You can see the results of the query in the Output directory in the file output_query1b.txt.

Task 2a

Assumptions
All tables should be in the format of 4 columns of integer.
When modifying the text file of the query1a, you can’t modify the letters, only the numbers.
The number of input should be less than 1800 input tuples. So we restrained the size of Q.

We implemented the single predicate IESelfJoin following the main ideas of the pseudo code of the paper but we decided to not use the permutation and bit array for this single predicate query. We just sorted the list as in the pseudo code and then we added the tuples that fits with the condition and then added them in an ArrayList. We had issues with Tuples so we decided to use ArrayList instead.

Results
You can see the results of the query in the Output directory in the file output_query2a.txt.

Task 2b

Assumptions
All tables should be in the format of 4 columns of integer.
When modifying the text file of the query1a, you can't modify the letters, only the numbers, and you can't modify the AND with a OR. We didn't notice where the problem was.
The number of input should be less than 1800 input tuples. So we restrained the size of Q.

We implemented the IE Self Join with two predicates, on the same method as the one explained in the paper of Khayyat Z. et al. "Fast and scalable inequality joins" VLDB Journal (VLDBJ), 2017, as asked in the instructions. Then, we use 2 ArrayList of Tuples (L1 and L2) from the data base which have been sorted resp. on their 2 columns appearing in conditions ,thank to the Sort class, in a direction which depend on the symbol of conditions. A permutation matrix contain the information to pass to L2 from L1.
In the core of the algorithm, we scan L1 from the beginning, we pass to 1 the byte associated to the current selected Tuple and scan L2 from the position in L2 (recorded thank to the matrix of permutation) of the current Tuple selected in L1. If Tuples scanned in L2 have a byte equal to 1 in the byte array, then that the first condition is checked (because have been already scanned in L1), we record the tuple because it implies that the second condition is checked too.

Way to improve our algorithm : optimize the part computing the matrix permutation, which is currently computed with 2 nested loops and a last one which scan all columns of the tuple to see if 2 tuples are equals.

Results
You can see the results of the query in the Output directory in the file output_query2a.txt.

Correctness
Also for this task we checked that the number of tuples using NLJ or IESelf were the same. However we realized that we don't find the same number of Tuples in some case, where there are some values which are equals and that the symbol of the condition is a superior strict or an inferior strict. One of the reasons is that when we scan L2, when it is a strict symbol, we jump the current value, to don't take the Tuple with itself, but we don't know if there isn't another Tuple with the same value… And in this case, we will do a mistake.

For instance, we tried with 1000 tuples as inputs. While for NLJ we obtained 244646 Tuples, we obtained 244824 output tuples.

Task 2c

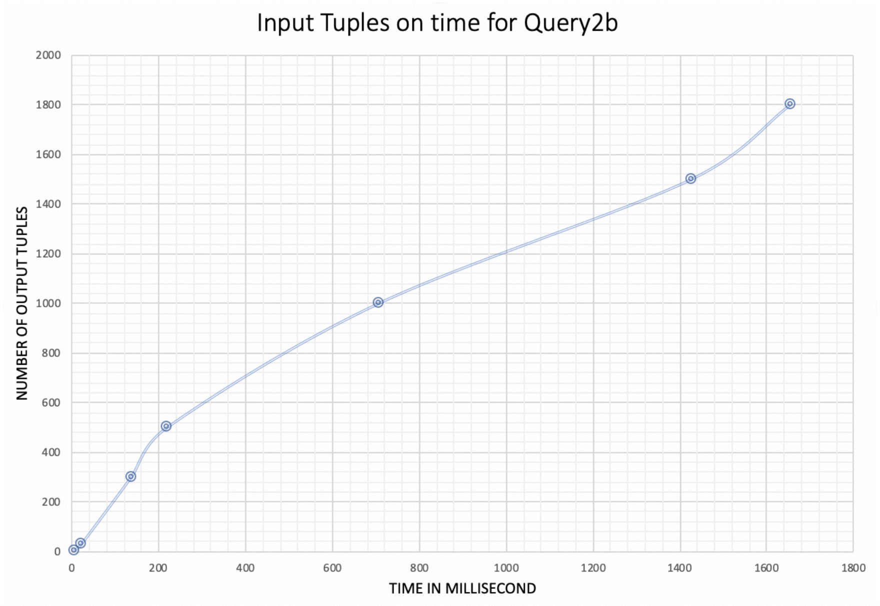
Assumptions
All tables should be in the format of 4 columns of integer.
When modifying the text file of the query1a, you can't modify the letters, only the numbers, and you can't modify the AND with a OR. We didn't notice where the problem was.
The number of input should be less than 1800 input tuples. So we restrained the size of Q.

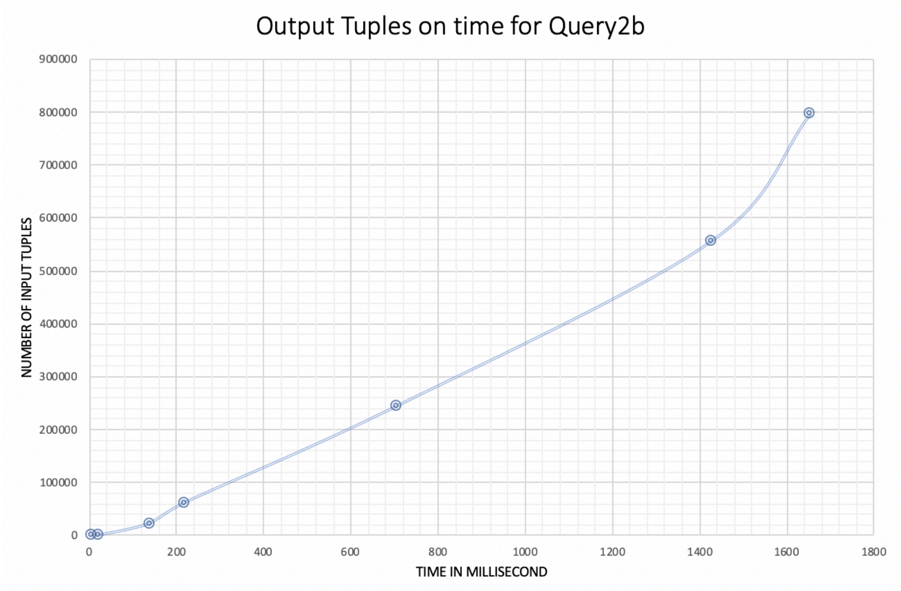
We implemented the IE Join with two predicates, on the same method as the one explained in the paper of Khayyat Z. et al. “Fast and scalable inequality joins” VLDB Journal (VLDBJ), 2017, as asked in the instructions. The method is similar to the IE Self Join, but we add 2 other ArrayList L1’ and L2’ which are similar to L1 and L2 but contain ordered Tuples from the other data base (other data base which didn’t exist in self join by definition). Also, we introduce 2 new ArrayList of int, O1 and O2 which are the matrix of « offset », and which contain the position in which the Tuple in L1 (resp. L2) would be if it was in L1’ (resp. L2’).

Results
You can see the results of the query in the Output directory in the file output_query2a.txt.

Plots for scalability

In order how scalable was our IESelf Join implementation, we plotted the number of Input Tuples according to time and the number of Output Tuples according to time for the Query2b example. We realized that somehow it cannot support more than 1800 tuples as input.


Input Tuples on time for Query2b
NUMBER OF OUTPUT TUPLES
TIME IN MILLISECOND


Output Tuples on time for Query2b
NUMBER OF INPUT TUPLES
TIME IN MILLISECOND