Database Management System Implementation
Minibase assignment 3: IEJoin

Group M:
Hugo DANET,
Victor MASIAK

1 GitHub repository

We hosted the files that we created / modified on a GitHub repository: `https://github.com/vim951/DBSys_lab3`

2 Outputs

The outputs correspond to the default queries, with 500 tuples loaded. They can be found in the `Output` folder of our git repository.

3 Summary

Task	**Status**
Task 1a	Implemented successfully
Task 1b	Implemented successfully
Task 2a	Implemented successfully
Task 2b	Implemented successfully
Task 2c	Very (very) close to a successful implementation
Task 2d	Not implemented

4 Global implementation

We have coded two functions (`GetTables` and `GetColumns`) allowing to create conditional expressions for simple predicate and double predicates from the txt query files. We have also implemented two functions allowing us to read the tables and the columns of the files. To test the implementations, you can uncomment the queries in the `runTests` function and indicate the path to the queries folder in the String constant path_to_queries.

`Q.txt`, `S.txt` and `R.txt` are loaded into the heapfile with the function `LoadDB` in `JoinTests.java`. Our implementations of 2a, 2b and 2c take FileScan iterators as an input. Then, we wrote algorithms 1 and 2 with ArrayLists, to ensure the proper executions of queries.

5 Task 1

We created two custom functions in the `Code/JoinTest.java` file, called Query7() (resp. Query8()) to answer understand better NestedLoopJoin.

Query7() execute the following query:

```
SELECT   B.bname
FROM     Sailors S, Boats B, Reserves R
WHERE    S.sid = R.sid AND R.bid = B.bid AND S.age >= 35
ORDER BY B.bname
```

Query8() execute the following query:

```
SELECT   B.bname
FROM     Sailors S, Boats B, Reserves R
WHERE    S.sid = R.sid AND R.bid = B.bid AND S.age >= 40 AND S.age <= 45
ORDER BY B.bname
```

Then we implemented `Query1a` and `Query1b`, which can handle simple predicate inequality join and double predicates inequality joins in .txt files. We first checked manually with a small number of tuples that the outputs were correct. Then, we coded a python function that takes the .txt files as input and returns the correct solution. We compared the solutions of our queries and of python with the bash command : `grep -vFf first_file.txt second_file.txt`

6 Task 2

We validated our classes by comparing the outputs of NLJ and our queries. It turned out that our classes work great, except 2c that does not exactly have the correct number of tuple returned.

In all cases, we tried to sort on Y first and then on X to take into account the following remark of the paper :

"One may observe that if the database contains duplicated values, when sorting one attribute X, its corresponding value in attribute Y should be considered, and vice versa, in order to preserve both orders for correct join result. Hence, in IESelfJoin, when sorting X, we use an algorithm that also takes Y as the secondary key."

However, those two successive sorts did not have the expected behavior, and despite our best efforts, we did not succeed in implementing our how sorting function on tuples.

6.1 Task 2a

We successfully adapted the algorithm 2 of the paper. We implemented it as a blocking operation, as it was easier to represent. We used the sort method for iterators, and then converted to arraylists.

6.2 Task 2b

Once again, we successfully implemented the algorithm 2 of the paper. We also did it as a blocking operation. We used the sort method for iterators, and then converted to arraylists.

6.3 Task 2c

We tried to implement algorithm 1 of the paper in our java class `IEJoinDoublePredicate`. The results are perfect for the Self Join Query 2c. However, we are not getting the correct number of tuples for 2c_1 and 2c_2 queries. We are very close, as it can be counted in a few dozen results on a query with 1000 times more outputs. The class performs well in terms of memory and speed.

7 Scalability

In terms of speed, our three classes are fully scalable. However, in terms of memory we cannot use our classes with more than 6500 tuples loaded in the heapfile. We don't know where the problem is coming from, maybe some limitations of our machines. This is probably also due to the fact that we used ArrayLists. We have made optimized versions of task 2a and 2b, which only load the columns of the joins. We get a better execution speed but we have the same limitations in terms of memory.

7.1 IESelfJoinSinglePredicate Performance on Query 2a

Number of tuples	100	500	1000	2000	5000
NLJ execution time	116 ms	1198 ms	2801 ms	9060 ms	40311 ms
IESJSP execution time	61 ms	133 ms	443 ms	2437 ms	11390 ms
Optimized IESJSP	38 ms	211 ms	574 ms	1626 ms	9704 ms
Number of outputs with NLJ	4950	124750	499500	1999000	12497500
Number of outputs with IESJSP	4950	124750	499500	1999000	12497500

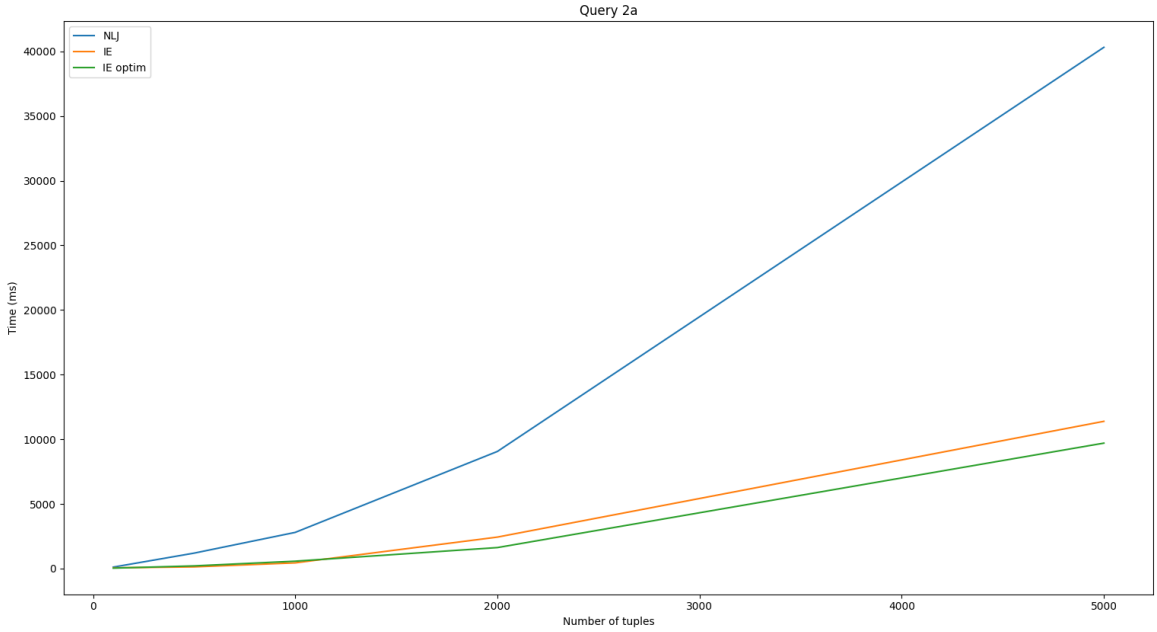

Figure 1: Query 2a execution time

7.2 IESelfJoinDoublePredicate Performance on Query 2b

Number of tuples	100	500	1000	2000	5000
NLJ execution time	226 ms	2004 ms	2978 ms	8824 ms	44168 ms
IESJDP execution time	87 ms	501 ms	820 ms	3923 ms	17018 ms
Optimized IESJDP	23 ms	205 ms	794 ms	2483 ms	15918 ms
Number of outputs with NLJ	4950	124750	499500	1999000	12497498
Number of outputs with IESJDP	4950	124750	499500	1999000	12497499

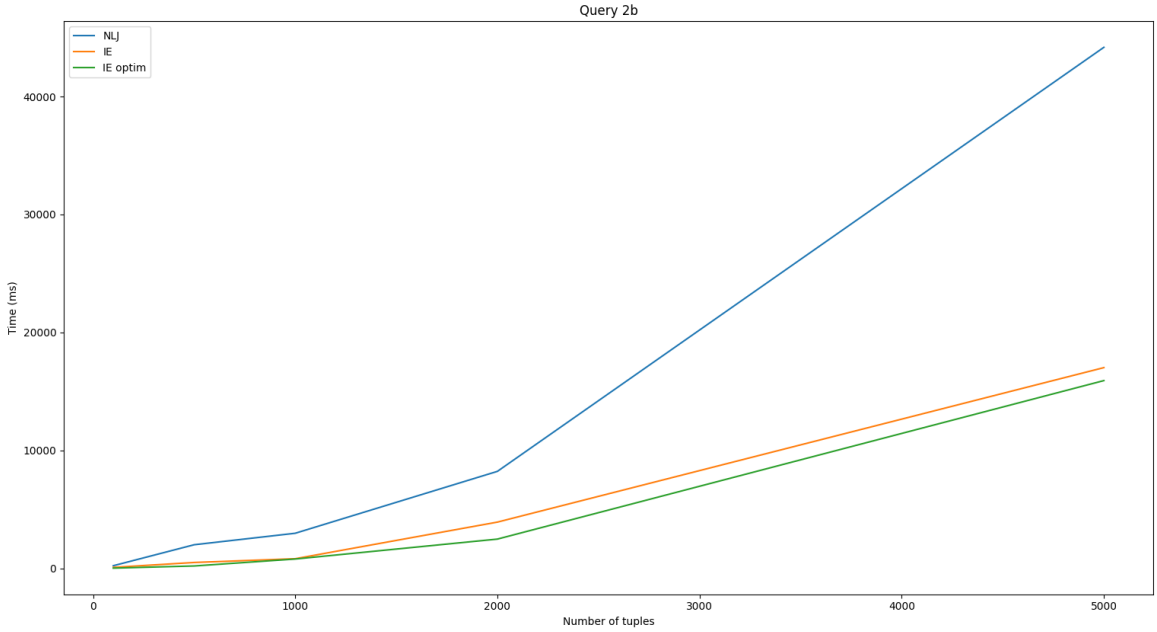

Figure 2: Query 2b execution time

7.3 IEJoinDoublePredicate Performance on Query2c

Number of tuples	100	500	1000	2000	5000
NLJ execution time	169 ms	1981 ms	3990 ms	10007 ms	49535 ms
IEDP execution time	119 ms	472 ms	1342 ms	4083 ms	26239 ms
Number of outputs with NLJ	4950	124750	499500	1999000	12497498
Number of outputs with IEDP	4950	124750	499500	1999000	12497498

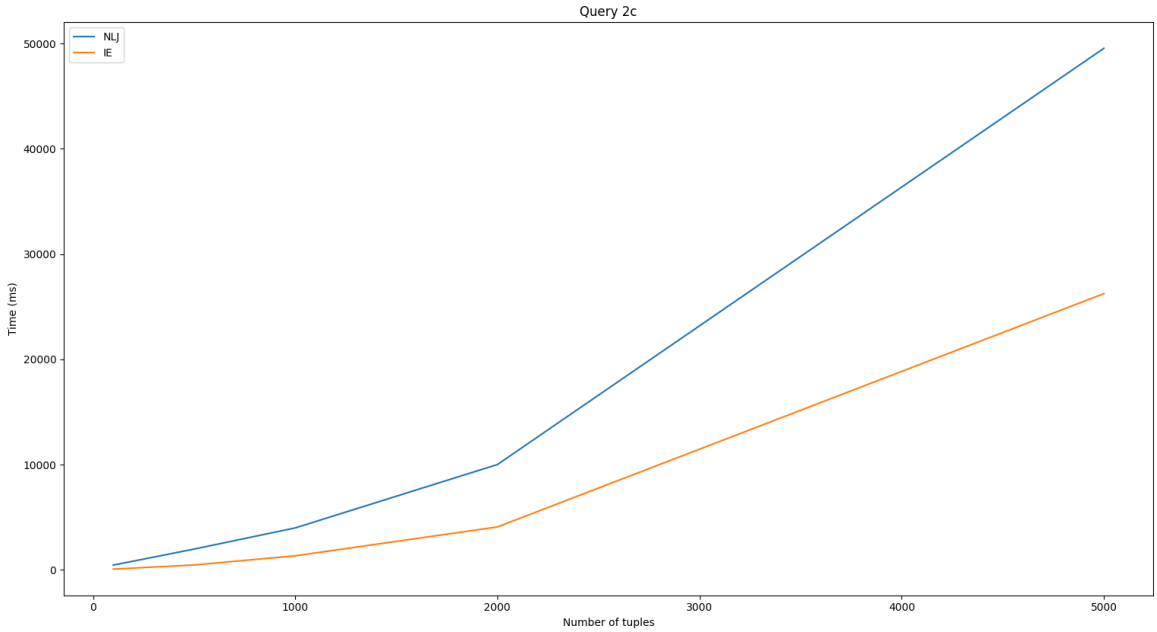

Figure 3: Query 2c execution time

7.4 IEJoinDoublePredicate Performance on Query2c_1

Number of tuples	100	500	1000	2000	5000
Number of outputs with NLJ	5099	30097	30097	30097	30097
Number of outputs with IEDP	5099	30128	30128	30128	30128

7.5 IEJoinDoublePredicate Performance on Query2c_2

Number of tuples	100	500	1000	2000	5000
NLJ execution time	228 ms	418 ms	529 ms	871 ms	2088 ms
IEDP execution time	61 ms	289 ms	2216 ms	2335 ms	12453 ms
Number of outputs with NLJ	5099	49132	99701	195437	488993
Number of outputs with IEDP	0	49196	99810	195646	489515

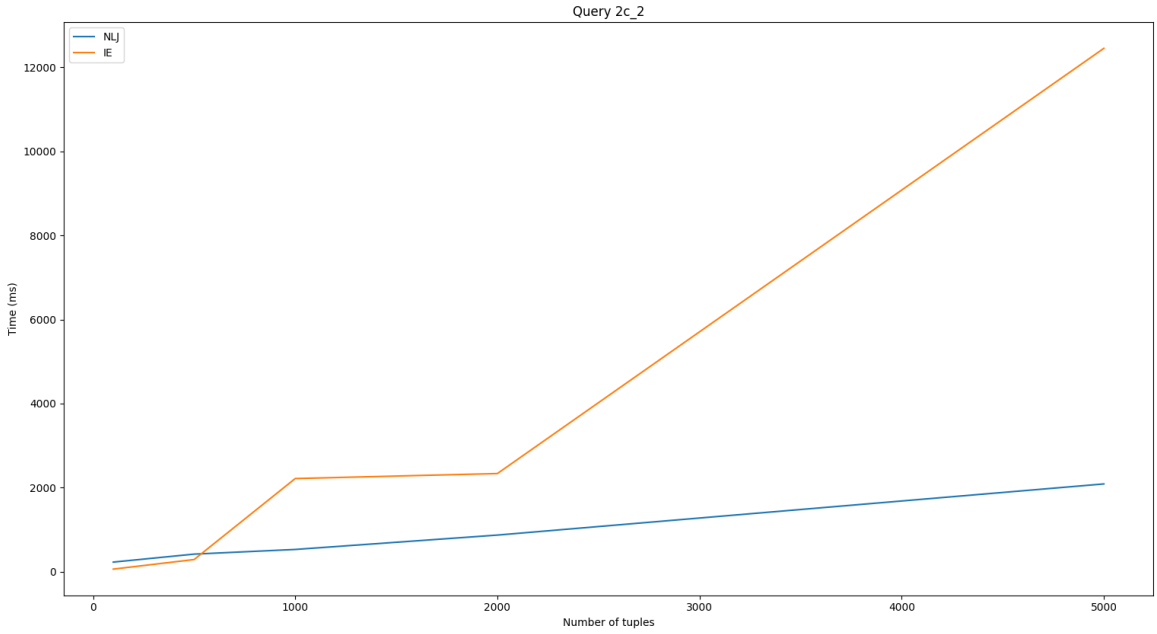

Figure 4: Query 2c_2 execution time

7.6 Interpretation

Quite logically, we can see that the single predicate self join is faster than the double predicate self join, which itself is faster than NLJ. The same can be said with the "normal" double predicate join of queries from 2c.

However, when it comes to the "normal" double predicate join of 2c_2, the plot is harder to discuss, but since the code does not fully work, it is probably normal.