CS440 - Assignment 3

Query Processing and Optimization

February 10, 2017

The assignment is to be turned in before Midnight (by 11:59pm) on February 7, 2017. You should turn in the solutions to this assignment as a pdf file through the TEACH website. The solutions should be produced using editing software programs, such as LaTeX or Word, otherwise they will not be graded.

Problem 1. Consider the natural join of the relation R and S on attribute A. Neither relations have any indexes built on them. Assume that R and S have 40000 and 20000 blocks, respectively. The cost of a join is the number of its block I/Os accesses. (3 points)

a. Assume that there are 50 buffer blocks available in the main memory. What is the fastest join algorithm to compute the join of R and S? What is the cost of this algorithm?

b. Assume that there are 300 buffer blocks available in the main memory. What is the cost of joining R and S using a sort-merge join? You should use a version of sort-merge algorithm that provides the minimum cost.

c. Assume that there are 202 buffer blocks available in the main memory. What is the cost of joining R and S using a sort-merge join? You should use a version of sort-merge algorithm that provides the minimum cost.

d. Assume that there are 150 buffer blocks available in the main memory. What is the fastest join algorithm to compute the join of R and S? What is the cost of this algorithm?

e. Assume that there are 80000 buffer blocks available in the main memory. What is the fastest join algorithm to compute the join of R and S? What is the cost of this algorithm?

a. Due to the small number of buffer pages, we can only use the improved block-based nested loop join algorithm or the sort-merge join algorithm with general multi-way merge sort. If you consider only sort-merge join algorithm with two-pass multi-way merge sort, improved block-based nested loop join is the fastest one. Otherwise, sort-merge join with general multi-way merge sort algorithm for sorting is the fastest algorithm. Both solutions are acceptable.

The cost of improved block-based nested loop join is $B(R) B(S) / M = 40000 \times 20000 / 50 = 16,000,000$ I/O accesses.
The cost of sort-merge join with general multi-way merge sort is

\[
\text{sorting} + 2B(R) + 2B(S)
\]

\[
= \left\lceil \log_{49} \left( \frac{40000}{50} \right) + 1 \right\rceil \times 2 \times 40000 - 40000 + \left\lceil \log_{49} \left( \frac{20000}{50} \right) + 1 \right\rceil \times 2 \times 20000 - 20000
\]

\[
+ 2 \times 40000 + 2 \times 20000
\]

\[
= 3 \times 2 \times 40000 - 40000 + 3 \times 2 \times 20000 - 20000 + 2 \times 40000 + 2 \times 20000
\]

\[
= 240000 - 40000 + 120000 - 20000 + 80000 + 40000
\]

\[
= 420,000 \text{ I/O accesses.}
\]

b. We have \( B(R) = 40000 \) and \( B(S) = 20000 \). As we have \( B(R) + B(S) < M^2 \), we may use the optimized sort-merge join. The cost of the join will be

\[
3 \times B(R) + 3 \times B(S) = 3 \times 40000 + 3 \times 20000 = 180,000 \text{ I/O accesses.}
\]

c. We have \( B(R) = 40000 \) and \( B(S) = 20000 \). As we have \( B(R) + B(S) > M^2 \), we cannot use the optimized sort-merge join. Because \( B(R) < M^2 \) and \( B(S) < M^2 \), we may use the (original) sort-merge join with two-pass multi-way merge sort. The cost of the join will be

\[
5 \times B(R) + 5 \times B(S) = 5 \times 40000 + 5 \times 20000 = 300,000 \text{ I/O accesses.}
\]

d. Because the size of \( R \) is larger than \( M^2 \), we cannot join \( R \) and \( S \) using sort-merge and optimized sort-merge algorithms with two-pass multi-way merge sort. However, the size of \( S \) is smaller than \( M^2 \), so we may use hash join algorithm to compute the join of \( R \) and \( S \). The cost of the join is

\[
3 \times B(R) + 3 \times B(S) = 3 \times 40000 + 3 \times 20000 = 180,000 \text{ I/O accesses.}
\]

We can also use block-based nested-loop join or sort-merge join with general multi-way merge sort, but their costs are higher than the one of the hash join algorithm.

e. Because both \( R \) and \( S \) fit in the main memory, we can use internal memory nested loop, sort-merge, or hash-join algorithms whose costs are

\[
B(R) + B(S) = 40000 + 20000 = 60000.
\]

Problem 2. Compare (improved) block-based nested-loop and sort-merge join algorithms for joining relations \( R \) and \( S \) in terms of their number of I/O accesses and memory requirements. Relations \( R \) and \( S \) do not fit in the main memory. Your analysis should compare these algorithms for both the case where each tuple of \( R \) joins with a few tuples of \( S \) and the case in which each tuple of \( R \) joins with many tuples of \( S \). (1 point)

The memory requirement in sort-merge join with two-pass multi-way merge sort is \( B(R) \leq M^2 \) and \( B(S) \leq M^2 \). On the other hand, the memory requirement in sort-merge join with general multi-way merge sort and (improved) block-based nested-loop join is \( M \).

In the case where each tuple of \( R \) joins with a few tuples of \( S \), then sort-merge join is better. If the memory requirement for sort-merge join with two-pass multi-way merge sort is satisfied, we should use such algorithm. If not, we should use sort-merge join with general multi-way merge sort.

On the other hand, if each tuple of \( R \) joins with many tuples of \( S \), then cost or sort-merge join may become \( B(R) \times B(S) + \text{sorting} \). In this case, (improved) block-based nested-loop join is better because it does not have to pay the cost of sorting.

Problem 3. Consider the following relations:
Product (name, production-year, rating, company-name)
Company (name, state, employee-num)

Assume each product is produced by just one company, whose name is mentioned in the company-name attribute of the Product relation. Attributes name are the primary key for relations Product and Company. Attribute company-name is a foreign key from relation Product to relation Company. Attribute rating shows how popular a product is and its values are between 1-5. The following statistics are available about the relations.

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(Product) = 45000</td>
<td>T(Company) = 500</td>
</tr>
<tr>
<td>V(Product, company-name) = 300</td>
<td></td>
</tr>
<tr>
<td>V(Product, rating) = 5</td>
<td></td>
</tr>
</tbody>
</table>

The following query returns the products with rating of 5 that are produced after 2000 and the states of their companies.

```
SELECT p.name, c.state
FROM Product p, Company c
WHERE p.company-name = c.name and p.production-year > 2000 and p.rating = 5
```

Suggest an optimized logical query plan for the above query. Then, estimate the size of each intermediate relation in your query plan. By an intermediate relation, we mean the relation created after each selection or join. (2 points)

The optimized logical query plan for the above query is the following. The plan first selects table Product using the conditions production-year > 2000 and rating = 5. It then projects Product on company-name and name. It also projects table Company on attributes name and state. Finally, it joins the projected Product and Company relations and projects the resulting table on attribute Product.name and Company.state.

There are two relations created after the selection and the join. Over the selection, the selectivity factor for point selection rating = 5 is 1/5, because V(Product, rating) = 5. The selectivity factor for range selection production-year > 2000 is 1/3 (magic number). Then, the size of the relation created after the selection over Product is 45000/(5 × 3) = 3000. For computing the size of join, we should have the number of distinct values for company-name for relation Company and Product after the selection. Because name is the primary key of relation Company, the number of distinct values of name in Company is 500. It is possible that after the selection, the number of distinct values of company-name in relation Product gets smaller than 300. However, because we assume that the attributes are independent in query optimization, it is reasonable to assume that the number of distinct values of company-name in the selected part of Product is still 300. Hence, the size of the relation created after the join is (3000 × 500) / max(300, 500) = 3000.

Projection operators do not change the number of rows in the input table.