Braden Ackles

CS 440

2/4/2017

Homework #3

1. 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/O accesses (3 points).
   1. 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?
      1. M=50, M2=2500
      2. Can't use Hash join B(R) > M2
      3. General Multi-way Merge: [O(B(R)\*logmB(R))+O(B(S)\*logmB(s))+2B(s)+2B(R)]= [O(40000\*log5040000)+O(20000\*log5020000)+2(20000)+2(40000)]=278980
      4. Improved Nested Loop: B(R)+[B(R)/(M-2)]\*B(S)=40000+[40000/50-2]\*20000=1.67E7
      5. The fastest cost being General Multi-way Merge: 278980
   2. 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 join? You should use a version of sort-merge algorithm that provides the minimum cost.
      1. M=300, M2=90000
      2. 2Pass: 3[B(R)+B(S)]+2b(S)+2B(R)= 3(40000+20000)+2(40000)+2(20000)=300000
      3. General Multi-way Merge: [O(B(R)\*logmB(R))+O(B(S)\*logmB(s))+2B(s)+2B(R)]= [O(40000\*log30040000)+O(20000\*log30020000)+2(20000)+2(40000)]=229039
      4. The fastest cost being General Multi-way Merge: 229039
   3. 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.
      1. M=202, M2=40804
      2. 2Pass: 3[B(R)+B(S)]+2b(S)+2B(R)= 3(40000+20000)+2(40000)+2(20000)=300000
      3. General Multi-way Merge: [O(B(R)\*logmB(R))+O(B(S)\*logmB(s))+2B(s)+2B(R)]= [O(40000\*log20240000)+O(20000\*log20220000)+2(20000)+2(40000)]=237263
      4. The fastest cost being General Multi-way Merge: 237263
   4. 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?
      1. M=150, M2=22500
      2. Can't use Hash B(R) > M2
      3. 2Pass: 3[B(R)+B(S)]+2b(S)+2B(R)= 3(40000+20000)+2(40000)+2(20000)=300000
      4. General Multi-way Merge: [O(B(R)\*logmB(R))+O(B(S)\*logmB(s))+2B(s)+2B(R)]= [O(40000\*log15040000)+O(20000\*log15020000)+2(20000)+2(40000)]=244123
      5. Improved Nested Loop: B(R)+[B(R)/(M-2)]\*B(S)=40000+[40000/150-2]\*20000=5.44E6
      6. The fastest cost being General Multi-way Merge: 244123
   5. 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?
      1. M=80000, M2=6400000000
      2. Hash join: 3(B(R)+B(S))=3(40000+20000)=180000
      3. General Multi-way Merge: [O(B(R)\*logmB(R))+O(B(S)\*logmB(s))+2B(s)+2B(R)]= [O(40000\*log8000040000)+O(20000\*log8000020000)+2(20000)+2(40000)]=175088
      4. Improved Nested Loop: B(R)+[B(R)/(M-2)]\*B(S)=40000+[40000/80000-2]\*20000=50000.3
      5. The fastest cost being Improved Nested Loop: 50000.3
2. Compare (improved) block-based nested-loop and sort-merge join algorithms for join- ing 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)

|  |  |
| --- | --- |
| Block Nested Loop | Sort-Merge Join |
| Block nested loop: This algorithm takes for all the blocks of S for each block in relation R and puts the in main memory one by one. This means that the amount of block accesses are (Br)\*(BS) + (Br)  Meaning:  For each block in Br of R  For each block in BS of S  For each tuple in Tr of Br  For each tuple in TS of BS  This will allow the output to return | Sort-Merge Join works by sorting the relations join attributes and then merging them. The number of block accesses(reads) is: Br +Br |
| Case: Each Tuple of R joins with a few tuples of S | Case: Each tuple of R joins with many tuples of S |
| Sort-merge would be better and more efficient here. It is better because each block has to be put in main memory one or possible more times but for complete efficiency it will only be put in main memory once. | Block Nested Loop is better for this case.  This is because if sort merge was doing it for each tuple of R joins with many tuples of S, the algorithm would need to read each block of S multiple times. There for since we already have to do this we might as well not do that and save the extra reads that would be required with sort-merge and use Block nested Loop. |

1. 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 companyname 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.

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)

|  |  |
| --- | --- |
|  | Tree sizes:  Tree 1: 45000/(5×3)=3000  Tree 2: (300×500)/500=300 |